UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)

Aplicación web para escritura colaborativa y estructurada

Autor:
Ismael Haddad

Ponente:
Antoni Urpí

Director:
Pontus Österberg

TRABAJO FINAL DE GRADO

Grado en Ingeniería Informática

Departamento de Ingeniería de Servicios y Sistemas de Información. ESSI

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)

13 de junio de 2019

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)

Resum

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
Departamento de Ingeniería de Servicios y Sistemas de Información. ESSI

Grado en Ingeniería Informática

Aplicación web para escritura colaborativa y estructurada

by Ismael Haddad

A la recerca d'una manera de compartir i desenvolupar idees amb un format ben organitzat i estructurat, sorgeix aquest projecte per oferir una plataforma on es puguin realitzar aquestes activitats. Com a objectiu es proposa oferir un espai per a l'exposició de discussions i gaudir d'un recopilatori de diversos tòpics per explorar.

Durant el projecte, s'especificaran els requisits i el disseny, i s'implementarà utilitzant tecnologies recents com *React Js*, *Ruby on Rails*, *Heroku*, etc. S'aplicaran metodologies *Agile* i es posarà en pràctica la gestió de projectes i la planificació.

Finalment, com a conclusió, es fa èmfasi en el desenvolupament de la intuïció de l'Enginyer de Software per adaptar-se a les eines que tingui a l'abast per solucionar qualsevol problema.

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)

Resumen

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
Departamento de Ingeniería de Servicios y Sistemas de Información. ESSI

Grado en Ingeniería Informática

Aplicación web para escritura colaborativa y estructurada

by Ismael Haddad

En busca de una manera de compartir y desarrollar ideas con un formato bien organizado y estructurado, surge este proyecto para ofrecer una plataforma donde se puedan realizar dichas actividades. Como objetivo se propone brindar un espació para la exposición de discusiones y disfrutar de un recopilatorio de varios tópicos para explorar.

Durante el proyecto, se especificarán los requisitos y el diseño, y se implementará utilizando tecnologías recientes como *React Js*, *Ruby on Rails*, *Heroku*, etc. Se aplicarán metodologías *Agile* y se pondrá en practica la gestión de proyectos y la planificación.

Finalmente, como conclusión, se hace énfasis en el desarrollo de la intuición del Ingeniero de Software para adaptarse a las herramientas que tenga al alcance para solucionar cualquier problema.

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)

Abstract

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
Departamento de Ingeniería de Servicios y Sistemas de Información. ESSI

Grado en Ingeniería Informática

Aplicación web para escritura colaborativa y estructurada

by Ismael Haddad

In search of a way to share and develop ideas with a well organized and structured format, this project arises to offer a platform where these activities can be carried out. The objective is to provide a space for the exhibition of discussions and enjoy a compilation of several topics to explore.

During the project, the system requirements and design will be described, and it will be implemented using recent technologies such as *React Js*, *Ruby on Rails*, *Heroku*, etc. *Agile* methodologies will be applied and project management and planning will be put into practice.

Finally, as a conclusion, emphasis is placed on the development of the software engineer's intuition to adapt to any tool available to solve any problem.

Índice general

Capítulo 1

Introducción

Para el Trabajo Final de Grado en la especialidad de Ingeniería del Software se hace este proyecto propuesto entre la empresa, Prototyp SE [1], y el estudiante, junto la Facultad de Informática de Barcelona de la Universidad Politécnica de Cataluña (UPC) mediante el convenio de prácticas externas en empresa (modalidad B), que consiste en la construcción de una solución software para el recopilatorio de diferentes ideas o temas de discusión y una manera de organizar las discusiones para que queden breves y precisas.

1.1. Contextualización

A la hora de hacer un ensayo de opinión, un artículo periodístico o una crítica literaria, de entre muchas formas de escritura, se han dado, a lo largo del tiempo, varias pautas para conseguir dicho fin. De estas pautas, para este proyecto, se ha querido destacar el desarrollo de ideas mediante el formato de discusión.

Hasta ahora, para conseguir una discusión estructurada, se han facilitado herramientas como las pautas para guiarse y las rúbricas para su valoración. Pautas y rúbricas, en las que se especifican varios aspectos que se deben tener en cuenta y que sirven como reglas del juego.

1.1.1. Marco teórico

A continuación se verá qué quiere decir cada término relacionado con el tema involucrado de una manera más clara y precisa, la teoría existente sobre la discusión de ideas y como una discusión bien organizada puede beneficiar al desarrollo de la misma.

El primer término que se explora es la *discusión de ideas*. Cuando se lleva a cabo, esta tiene como fin el conciliar las partes de las ideas que sean ciertas y deshacerse de las que no lo sean. La *discusión de ideas* forma parte de muchos conceptos que se engloban en el significado de *pensamiento crítico*. El *pensamiento crítico*, se define, basada en encuestas realizada por profesores y estudiantes, como:

> «*La actividad central del pensamiento crítico es la evaluación de lo que podría llamarse evidencia para hacer un juicio.*»
>
> -Moon, 2008 [2]

Es decir, evaluar una premisa que justifica una conclusión.

Un *argumento* básico, según Moore y Parker, está compuesto por una *premisa* y una

conclusión. Y una *premisa* es el razonamiento o prueba que se da para apoyar una *conclusión* [3].

Para conseguir un modelo de argumento más preciso, se toma como referencia el método Toulmin. Este, propone que un argumento está formado por la *conclusión*, que vendría a ser la afirmación con la cual partimos; los *datos*, son las razones que se dan; la *garantía*, el cómo se llega a los datos; y el *respaldo*, la demostración de la validez de la garantía [4].

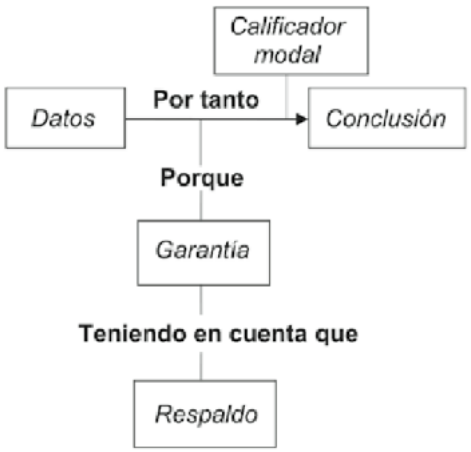


FIGURA 1.1: Modelo de argumentación de Toulmin

Por otro lado, según el siguiente estudio sobre la construcción de conocimiento en foros virtuales de discusión, en contraste con los foros no estructurados, los foros estructurados a base de reglas para el desarrollo de la argumentación tiende a elevar la calidad de la discusión. Se fomentaba la *"comparación de la información, la negociación de significados, la síntesis y la aplicación de nuevos significados"* [5].

Viendo lo anterior, se plantea construir un sistema moderno que ofrezca una forma de llevar estas discusiones estructuradas gracias a las recientes tecnologías web, como las páginas dinámicas, que desde el 1997 ha permitido añadir nuevas maneras de interactuar con las páginas web, y los *WebSockets* [6] que, desde el 2011, ha ofrecido una manera de interactuar con el sistema en tiempo real, hasta tal punto que, juntando estas dos, lo que se obtiene es una aplicación con capacidad completa de satisfacer cualquier requisito.

1.1.2. Motivación

Uno de los intereses de este proyecto es poder tener una herramienta que permita controlar lo mejor posible el proceso de argumentación. Siguiendo las pautas del *Critical Thinking* de Moore y Parker, nace la necesidad de un método que ofrezca una ayuda a la hora de plantear los argumentos. Tradicionalmente, se ha apoyado en rúbricas [7], pero se cree que se pueda implementar una adaptación con las nuevas tecnologías y hacer dicho proceso más cómodo.

Otro motivo de este proyecto, surge a partir de la observación de los *hosts* de sistemas de control de versiones, tales como *Github* [8] o *Bitbucket* [9], donde los usuarios

no solo lo usan como un sitio para la colaboración de un equipo en un determinado proyecto de software, sino que también se comparte estos códigos al resto del mundo para fines como el aprendizaje, el compartir módulos para la integración en otros software y, en general, tener una amplia biblioteca para tener a mano a la hora de construir otros programas software.

Entonces, partiendo de esta dinámica y queriendo enfocar para el ámbito de la escritura, se quiere conseguir una herramienta donde se facilite la recopilación de diferentes ideas, argumentos o discusiones para que, de forma análoga al desarrollo de software, se puedan compartir y explorar y, así, asistir a la creación de ensayos, artículos periodísticos, críticas literarias o cualquier escrito en general.

1.2. Resumen de intenciones para el proyecto

En breve, esta aplicación web ofrecerá una manera de organizar las ideas de los escritores en forma de conversaciones o debates. A diferencia de los foros o mediante papel y pluma, nos permitirá crear avatares con diferentes opiniones, los cuales argumentarán para defenderlas (Fig. 1.2). Estos avatares pueden estar asignados todos a un único usuario para facilitar el desarrollo de las diversas ideas en un formato de debate o se pueden asignar a otros usuarios y aprovechar la funcionalidad colaborativa. Habrá un límite de 100 palabras por argumento, lo que permitirá una expresión clara y breve de la idea. Y habrá una sección de comentarios y valoraciones.

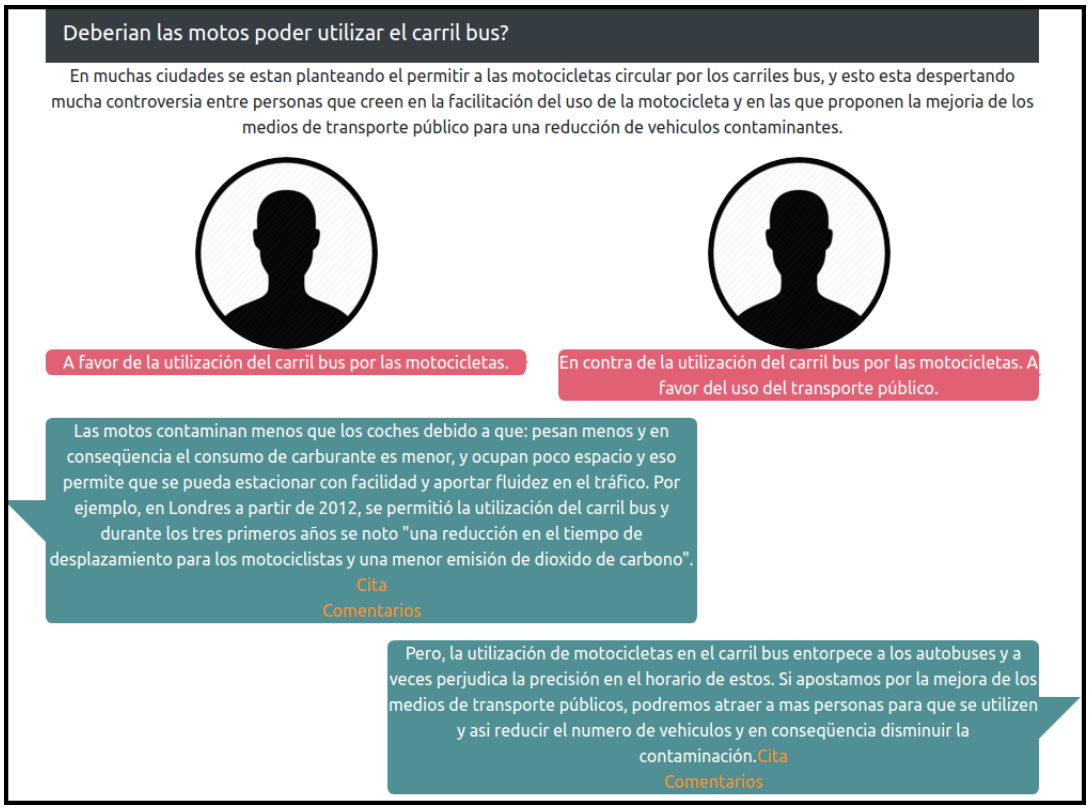


FIGURA 1.2: Mock up de la pantalla de discusión

Capítulo 2

Estado del arte

Un estudio de mercado consiste en buscar y analizar las soluciones existentes que comparten parcialmente o completamente los objetivos del proyecto. Es importante hacer dicho estudio porque permite conocer la competencia y, de ahí, aprender de sus puntos fuertes y débiles. La idea es no construir algo ya existente, si no, enfocar el proyecto en aportar soluciones nuevas y mejorar las obsoletas.

2.1. Estudio de mercado

Para buscar las diversas alternativas que se ofrecen en el mercado, se han tomado algunos criterios como los siguientes puntos:

- ¿Ofrece un espacio para la discusión de ideas?
- ¿Se da las herramientas para dar apoyo a la hora de argumentar?
- ¿Hay discusiones en privado y una manera de desarrollar dichas discusiones con un solo usuario?
- ¿Ofrece un repertorio de tópicos de discusión?

A continuación se observa las diferentes aplicaciones encontradas:

2.1.1. *Debate.org*

Esta aplicación web ofrece tres secciones o formas de discusiones: los debates, las opiniones y el foro. Los debates consiste en un uno-contra-uno dividido en rondas, donde otros usuarios votan por quien ha ganado el debate. En la sección de opiniones, se plantea una cuestión y los usuarios votan a favor o en contra y dejando una opinión. Y el foro se trata de un foro normal y corriente.

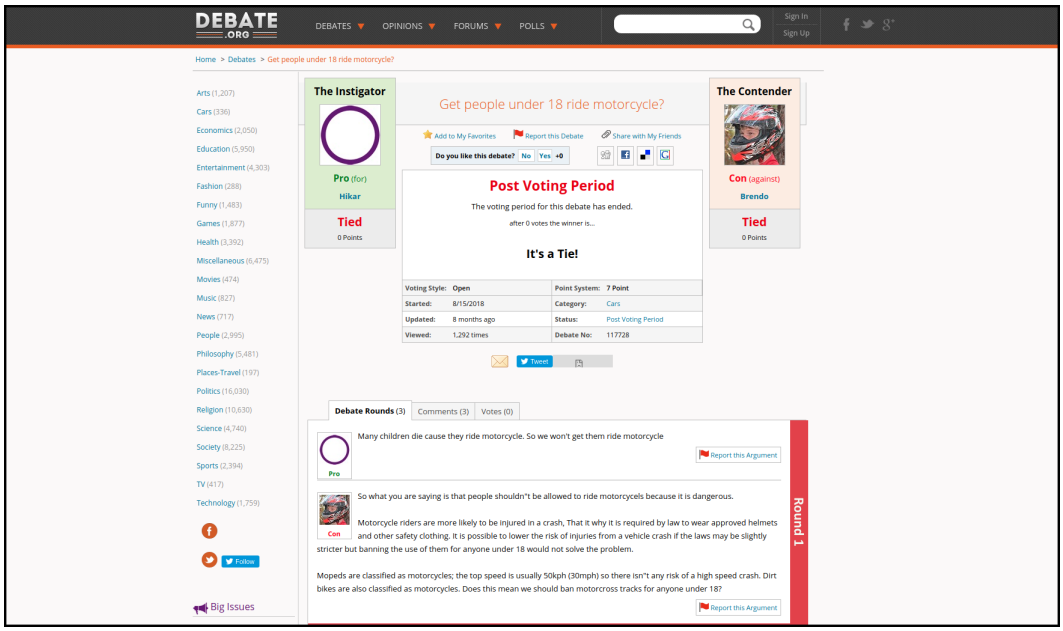


FIGURA 2.1: Captura de pantalla de *Debate.org*

2.1.2. *Slack*

Slack es una aplicación enfocada en la comunicación en una empresa que destaca en su organización de las conversaciones, divididas en canales donde hay participantes asignados y ofrece la capacidad de crear hilos a partir de las conversaciones y de compartir archivos fácilmente.

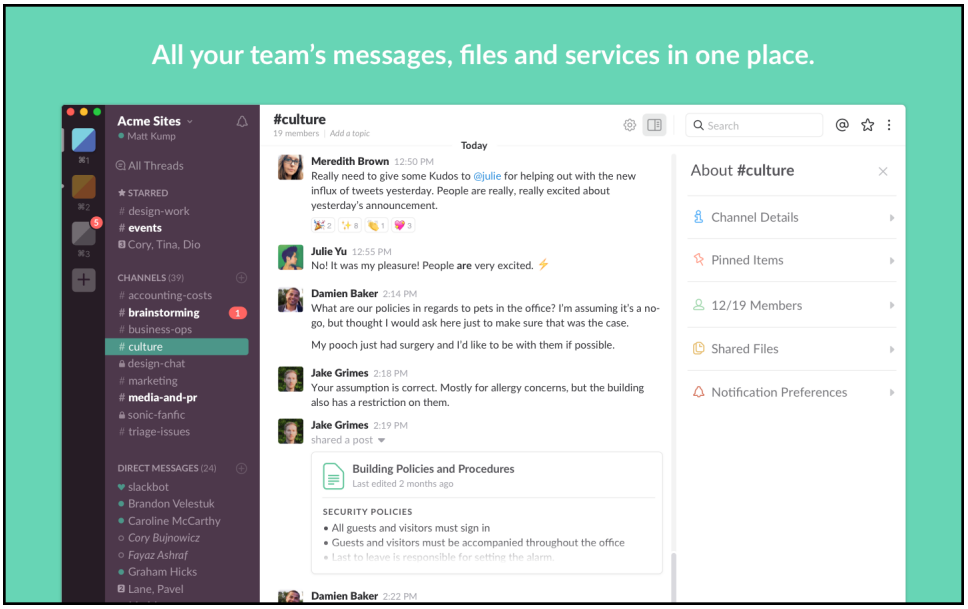


FIGURA 2.2: Captura de pantalla de *Slack*

2.2. Conclusión del estudio de mercado

Esta aplicación destaca en la insistencia de la construcción de un argumento breve, claro y preciso, gracias a la limitación por palabras y el sistema de rúbrica que dará

asistencia al usuario a la hora de publicar un argumento. También se evita el ganar o perder un debate, ya que se cree que el fin de las discusiones que se quiere ofrecer son para el desarrollo de nuestras propias ideas. Y, además de poder tener discusiones privadas y poderlas desarrollar por nuestra cuenta, también tenemos la posibilidad de hacer una copia de una discusión pública y luego poder asignar los avatares a otros usuarios y continuar con dicha discusión.

	Debate.org	**Slack**	Aplicación a desarrollar
Ofrece un espacio para la discusión de ideas	SI	SI	SI
Se da las herramientas para dar apoyo a la hora de argumentar	NO	NO	SI
Límite de palabras en un bloque de argumento	NO	NO	SI
Limite de tiempo y ganador del debate	SI	NO	NO
Discusiones en privado y modo “solitario”	NO	SI	SI
Repertorio de tópicos de discusión	SI	NO	SI
Hacer copias de discusiones públicas y poder continuarlas	NO	NO	SI

CUADRO 2.1: Tabla de comparación de las alternativas en el mercado

Capítulo 3

Objetivos y alcance del proyecto

A continuación, se exponen los objetivos propuestos para este proyecto y el alcance que tendrá al principio y al final.

3.1. Objetivos

Este proyecto consiste en desarrollar una plataforma web que pretende dar a los usuarios una forma de gestionar, guardar y organizar diferentes temas, tópicos o ideas desarrolladas mediante un formato de discusión. Se empezará con un alcance bien definido y se tendrá la posibilidad de ampliar el alcance desde el comienzo.

En primera instancia, la aplicación ofrecerá una función para la discusión entre diferentes ideas.

En segunda, se podrá organizar cada uno de los temas, pudiendo compartir discusiones propias o clonar y retomar discusiones de otros.

El objetivo será tener una plataforma web que ofrecerá una manera de almacenar y organizar un repertorio de ideas para que el escritor tenga a mano a la hora de hacer sus borradores.

3.2. Alcance

El alcance del proyecto está, inicialmente, delimitado por dos grandes módulos: La discusión y la biblioteca de tópicos.

La discusión estará formada por una pantalla donde se podrá ver el título y la descripción del tópico a discutir, dos avatares con las opiniones a defender y los usuarios asignados para tomar los respectivos roles, los argumentos propuestos con su sección de comentarios para cada uno y la valoración, y finalmente una tabla con puntos en acuerdo y en desacuerdo. Las funcionalidades que ofrecerá esta pantalla son la de enviar un argumento de un máximo de 100 palabras, proponer un punto de concordancia, aceptar o rechazar dicho punto, invitar alguien mediante *email* a participar en la discusión y asignar un avatar a un determinado participante.

La biblioteca de tópicos contendrá una pantalla para gestionar nuestra lista de tópicos y otra para explorar los tópicos públicos de otros usuarios. Las funcionalidades básicas que ofrecerán son la de crear una nueva discusión, editar o borrar una discusión propia y poder clonar o hacer *fork* a una discusión ajena.

Como funcionalidades extras, en el caso de contar con más tiempo de lo previsto, tenemos:

- Un sistema para añadir citas en los argumentos, para no gastar las escasas palabras que disponemos.
- Relacionar un bloque de argumento como respuesta a un otro.
- Un sistema de rúbrica que guíe al usuario en el proceso de argumentación.

3.3. Posibles obstáculos y riesgos

Según la regla de *Pareto*, usada en la gestión empresarial, se dice que un 80 % del valor de un producto software viene de un solo 20 % de las funcionalidades que ofrece y el otro 80 % de funcionalidades solo aportan un 20 % del valor. Para reducir al máximo el riesgo por falta de tiempo, se enfocará en realizar las funcionalidades esenciales y que más valor aporta al proyecto para que este tenga entidad suficiente para ser un Trabajo Final de Grado.

Al utilizar una metodología *Agile*, que consiste en una lista de funcionalidades pendientes priorizadas, donde estas son independientes entre ellas, esta ofrece bastante flexibilidad cuando surge cualquier imprevisto. Se puede implementar un plan alternativo y tomar las consideraciones oportunas al momento.

Como ultimo punto, se debe mencionar como posible obstáculo el desconocimiento de ciertas tecnologías y los posibles errores que puedan surgir a lo largo del desarrollo del proyecto. Como medida para ello, se dispone de una persona, el *Senior* de la empresa, que se ofrecerá para posibles dudas y aconsejar en aquello que este relacionado con la tecnología implicada.

Capítulo 4

Metodología y rigor

A continuación, se expone el proceso a seguir y las diferentes herramientas involucradas.

4.1. Proceso a seguir

Para el desarrollo de este proyecto se utiliza la metodología *Agile*, ya que es la manera en que se trabaja en esta empresa. En este proyecto en particular, se seguirá el método *Scrum*, donde se tendrá un *Backlog*, es decir, una lista priorizada de tareas pendientes. Estas tareas, serán los requisitos del sistema y estarán definidas como *historias de usuario*.

Habrá varias iteraciones de dos semanas de durada cada una, donde hay una entrega o *release* en cada final. La *release* consiste en un despliegue del código a producción y tendrá forma de producto completo, donde cada iteración tendrá una nueva mejora. En cada iteración hay una reunión con el director de la empresa para hacer la retrospectiva y otra con el *Senior* de la empresa para la revisión de código.

Durante la planificación inicial, se hará un análisis de los requisitos que tendrá la aplicación a desarrollar con *Pivotal Tracker* [10] y *Plan IT Poker* [11]. Se documentará el TFG en *Google Docs* [12]. Y se comunicará con el director de la empresa y el ponente del TFG mediante correo electrónico.

Una iteración consistirá en asignar las funcionalidades prioritarias a desarrollar con *Pivotal Tracker* [10] al comienzo. Documentar los objetivos, diseñar, implementar y testear el código. Y, finalmente, documentar los resultados y hacer el despliegue de la aplicación con *Travis-CI* [13], en los respectivos servidores [8] [14] [15].

4.2. Herramientas de seguimiento

Para gestionar las *historias de usuario*, se utilizará *Pivotal Tracker* [10], donde se priorizan y se asignan puntos de carga de trabajo, y la aplicación se encarga de repartir las *historias de usuario* a través de las iteraciones.

Los puntos de carga de trabajo se definirán con la ayuda de *Plan IT Poker* [11].

Mediante *Google Docs* [12], se irá documentando todos los aspectos relacionados con el TFG.

Github [8] y *Git* [16], permitirá seguir el historial de desarrollo del código.

4.3. Herramientas de validación

Travis-CI [13] probará el código cada vez que se añada un cambio al producto final y automáticamente desplegará dicho cambio a producción.

Mini Tests [17] es la librería que permitirá escribir las pruebas de calidad para evitar que el código no se estropee a medida que se vaya desarrollando.

Capítulo 5

Planificación temporal

Se divide este capitulo en la planificación inicial y las desviaciones ocurridas.

5.1. Planificación inicial

A continuación se expone la duración del proyecto, los recursos disponibles, una descripción de las tareas implicadas y como se asocia todo para construir el diagrama de *Gantt* [18].

5.1.1. Duración del proyecto estimado

La duración del convenio de prácticas externas comprende desde el 4 de febrero y finalizando el 17 de junio. La carga de trabajo que se exige es de 735 horas, de las cuales 590 horas, se dedicarán al proyecto del TFG. El resto se dedicará a otros proyectos similares que vayan surgiendo durante la estancia en la empresa para adquirir las competencias exigidas por el convenio.

5.1.2. Recursos

De los recursos para el proyecto, diferenciamos los personales, los materiales y los software.

De personales, tenemos a 4 personas:

- El **autor del TFG**, que se dedicará a la gestión y la planificación del proyecto, el análisis de los requisitos, el diseño y la implementación del sistema software, y la documentación. Dedicará 40 horas semanales. Los roles que asumirá son: Jefe de proyecto, Arquitecto, Diseñador, Programador y *Tester*.
- El **director de la empresa**. Verificará que el proyecto vaya en buen camino, aportando nuevas ideas o consejos para mejorarlo. Se dedicará 1 hora aprox. semanal para la reunión de la retrospectiva.
- El ***Senior* de la empresa**. Verificará que el código esté correcto y responderá a dudas para mejorar el código. Se dedicará 1 hora aprox. semanal para la reunión de la revisión del código.
- El **Ponente del TFG**. Guiará el estudiante durante el transcurso del TFG y resolverá cualquier tipo de dudas que tenga relación con ello. Se comunicará mediante correo electrónico y habrá reuniones cuando convenga.

De materiales, tenemos:

- Lugar de trabajo. Tendrá un ambiente que facilite llevar a cabo el trabajo cómodamente.
- Ordenador portátil *Toshiba Satellite P50* [19]. Tendrá *Ubuntu* [20] como sistema operativo donde se facilitará las varias aplicaciones necesarias para llevar a cabo el proyecto (editor de texto, navegador, etc).
- Servidor *Google Drive* [12]. Es donde la documentación y cualquier material, se almacenará.
- Servidor *Github* [8]. Es donde el código en desarrollo residirá.
- Servidores *Heroku* [14] y *Netlify* [15]. Es donde el código a producción residirá.

De software, tenemos:

- *Ubuntu 16.04 LTS* [20]. Una distribución del sistema operativo *Linux* [21].
- *Google Chromium* [22]. Navegador web open source.
- *Atom* [23]. Editor de texto open source.
- *Git* [16]. Sistema de control de versiones de código open source.
- *Postman* [24]. Aplicación para testear el software.

5.1.3. Plan de acción y valoración de alternativas

En la fase de implementación, habrá variabilidad en cuanto a número de funcionalidades que entran en cada iteración. Si algunas funcionalidades se acaban implementado antes de su tiempo estimado, entonces se podrá mover funcionalidades previstas para la siguiente iteración a la actual. Y si, por lo contrario, se tarda más de la cuenta, se podrá mover para la siguiente. También hay que tener en cuenta que cada funcionalidad están pensadas para ser lo más independiente posible del resto de funcionalidades. Esto nos permite mucha flexibilidad a la hora de optar por diferentes alternativas a la mitad del proyecto.

También, a medida que el proyecto avance, se irá recopilando una lista de funcionalidades extras que entrarían dentro del alcance si nos quedamos con más tiempo de lo previsto.

Con una posibilidad media, se podría dar que no tengamos alguna funcionalidad principal al acabar todas las iteraciones, entonces se hará una iteración adicional.

5.1.4. A tomar en cuenta

Esta aplicación, al estar claramente definida en diferentes módulos y funcionalidades, se puede paralelizar el desarrollo de la misma de manera muy eficaz. Pero, como solo tendremos, como recursos personales, a una única persona, esto se hará secuencialmente.

5.1.5. Descripción de las fases y tareas

El proyecto estará dividido en las siguientes 4 fases:

Fase planificación inicial

En esta primera fase se deciden los objetivos del proyecto y se formulan los primeros requisitos que tendrá el sistema en forma de *historias de usuario* priorizados en un *Backlog*. Para la planificación de la implementación, cada *historia de usuario* tendrá unos puntos de carga de trabajo y se repartirán entre las iteraciones. También se dedicará al aprendizaje previo de las tecnologías implicadas y poner en marcha el entorno de trabajo.

Fase de Gestión de Proyecto (GEP)

Esta fase empieza el 18 de febrero y acaba el 30 de marzo, y consiste en la asignatura de GEP, donde se entregará cada semana determinados apartados que formarán parte de la documentación final. Se estudiará el material provisto y se redactará las entregas.

Fase de implementación

Está dividido en 6 iteraciones, de dos semanas cada una, que consisten en:

1. Asignar las funcionalidades a implementar para la actual iteración.
2. Documentar los objetivos de la iteración.
3. Diseñar e implementar las funcionalidades.
4. Hacer *tests* para cerciorarse de que el código sigue funcionando y que sea de buena calidad.
5. Documentar los resultados de la iteración.
6. Hacer una nueva *release*, que consiste en desplegar el código a producción.

Fase final y cierre

Finalmente, se acabará con la documentación, puliendo la memoria, y preparando la defensa del TFG en los tribunales. También, se dedicará tiempo para los últimos retoques de la aplicación.

5.1.6. Calendario

Estimación de las horas

En la siguiente tabla podemos ver las horas previstas para cada tarea. Hay que tener en cuenta que las fechas de la fase de implementación de las 3 primeras iteraciones y la fase de gestión de proyecto se solapan, y en consecuencia, en estas iteraciones se dedicará menos tiempo que el resto.

Tareas	**Responsable**	**Horas**
Fase planificación inicial		**80.0**
Definir objetivos del proyecto	Jefe de proyecto	20.0
Aprendizaje previo	Jefe de proyecto/Programador	40.0
Config. del entorno de trabajo	Jefe de proyecto	20.0
Fase de Gestión de Proyecto		**75.0**
Entrega 1: Contexto y Alcance	Jefe de proyecto	25.0
Entrega 2: Planificación temporal	Jefe de proyecto	8.0
Entrega 3: Gestión econ. y Sosteni.	Jefe de proyecto	9.0
Entrega 4: Documento completo	Jefe de proyecto	18.0
Preparación y presentación oral	Jefe de proyecto	15.0
Fase de implementación (3 primeras iteraciones)		**55 x 3**
Definir objetivos de la iteración	Jefe de proyecto	5.0
Diseñar base de datos	Arquitecto	10.0
Implementar backend	Programador	10.0
Diseñar interfaz de usuario	Diseñador	10.0
Implementar frontend	Programador	10.0
Testear	Tester	3.0
Documentar resultados	Jefe de proyecto	5.0
Reunión de control	Jefe de proyecto, Director empresa	1.0
Reunión de revisión de código	Jefe de proyecto, Senior empresa	1.0
Fase de implementación (resto de iteraciones)		**80 x 3**
Definir objetivos de la iteración	Jefe de proyecto	5.0
Diseñar base de datos	Arquitecto	10.0
Implementar backend	Programador	15.0
Diseñar interfaz de usuario	Diseñador	10.0
Implementar frontend	Programador	20.0
Testear	Tester	13.0
Documentar resultados	Jefe de proyecto	5.0
Reunión de control	Jefe de proyecto, Director empresa	1.0
Reunión de revisión de código	Jefe de proyecto, Senior empresa	1.0
Fase final y cierre		**30.0**
Redacción memoria y documentación	Jefe de proyecto	15.0
Preparación Defensa TFG	Jefe de proyecto	15.0
TOTAL		**590.0**

CUADRO 5.1: Estimación de horas

Diagrama de *Gantt* [18]

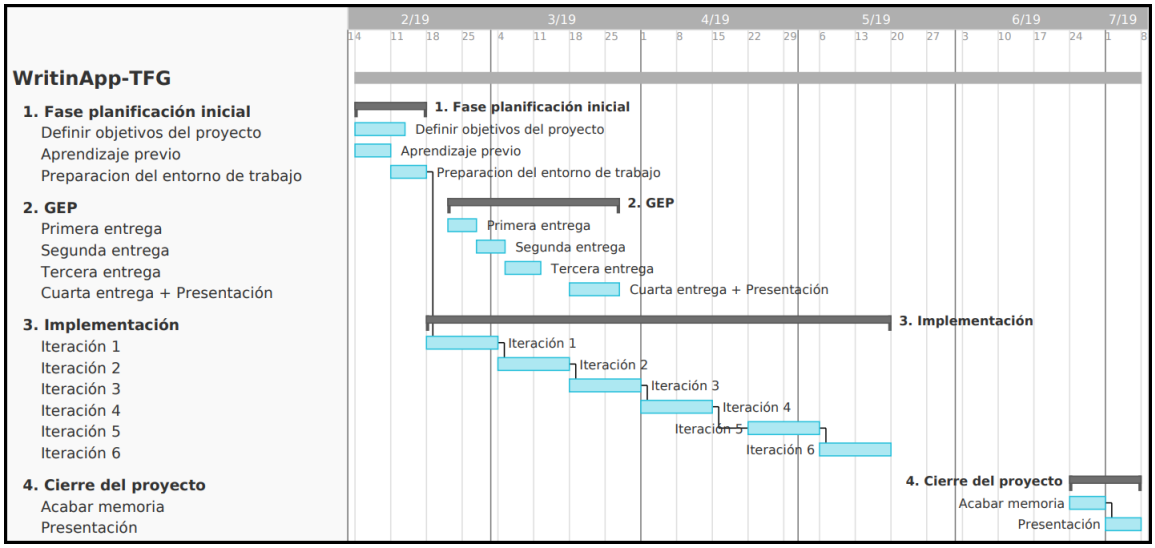


FIGURA 5.1: Diagrama de *Gantt*

5.2. Retrospectiva sobre la planificación

A continuación se expone los cambios hechos en contraste al plan original y las conclusiones sacadas.

5.2.1. Cambios del plan original

En un principio, se tenía pensado hacer la sexta iteración para implementar nuevas funcionalidades, pero se ha visto que el proyecto empezaba a acercarse al punto de rendimiento decreciente, es decir, cada funcionalidad extra aportaba poco valor y aumentaba la complejidad del proyecto. Entonces, se ha decidido utilizar esta iteración para pulir y dar los últimos retoques a la aplicación final.

Esto no tiene efecto en cuanto a tiempo y costes, pero sí al alcance del proyecto. Pero, como se ha dicho anteriormente, ya que son funcionalidades extras que no aportan mucho valor, esta modificación no afecta negativamente a los objetivos del proyecto, y se podría decir que se mejora la calidad del producto final al prescindir de estas.

Las funcionalidades de las cuales se han prescindido son:

- Añadir citas en los argumentos mediante una sección adicional.
- Referenciar bloques de argumentos para responder.

5.2.2. Conclusiones

Al final, se puede decir que el proyecto ha cumplido la planificación estimada y esto es atribuible a la flexibilidad que ofrece la propia metodología *Agile*. Cuando ha faltado tiempo, se ha replanificado fácilmente las funcionalidades a implementar para retomarlas en la siguiente iteración. Y cuando ha sobrado, se ha adelantado trabajo de la siguiente.

En cuanto a la asimilación de los objetivos, se han cumplido las funcionalidades fundamentales de la aplicación, como la discusión y la lista de tópicos, y se ha podido hacer algunos de los extras, como las valoraciones y los comentarios. Prescindir del resto de funcionalidades ha sido debido a otros factores, tales como el valor aportado al proyecto y el aumento de complejidad innecesaria, y no por falta de tiempo por una mala planificación.

Capítulo 6

Gestión económica y sostenibilidad

A continuación, se exponen la gestión económica y las reflexiones sobre las tres dimensiones de la sostenibilidad.

6.1. Autoevaluación sobre la sostenibilidad

Después de hacer el cuestionario sobre la sostenibilidad en las TIC, se ha aprendido la importancia de la literatura existente sobre la problemática y las soluciones de las diferentes dimensiones. Para cada dimensión, se describen diversos indicadores para controlar los diferentes problemas e identificar las posibles causas, y se recopila varias soluciones para afrontarlas.

De las cosas que tengo que mejorar a la hora de valorar la sostenibilidad y sus 3 dimensiones, la económica, la ambiental y la social, son las siguientes:

Sobre la económica, hay que tomar en cuenta que el proyecto sea viable. Para ello tenemos que estimar los costes de todos los recursos implicados y prevenir lo mejor posible los riesgos que puedan producirse. Sobre la ambiental, el objetivo es reducir los recursos ambientales o huella ecológica. Una manera sencilla de medir estos recursos es calcular el consumo eléctrico. Y de posibles soluciones está la reutilización de recursos. Sobre la social, se reflexiona él cómo puede aportar este proyecto a nivel personal o a nivel comunitario, teniendo en cuenta la ética. El objetivo es mejorar la calidad de vida de la sociedad.

6.2. Costes del proyecto

A continuación veremos que recursos son necesarios para llevar a cabo el proyecto y, a partir de ello, haremos un cálculo de los costes que tendrá este proyecto, teniendo en cuenta los costes directos e indirectos, las contingencias y los riesgos.

6.2.1. Recursos necesarios

Tenemos 3 tipos de recursos: humano, material y software.

	Roles / Concepto	Descripción
Recurso humano	Jefe de proyecto	Encargado de liderar, gestionar y documentar el proyecto.
	Arquitecto	Encargado del diseño de la base de datos.
	Diseñador	Encargado del diseño de la interfaz de usuario.
	Programador	Encargado de la implementación del sistema.
	Tester	Encargado de las pruebas de calidad del código.
Recurso material	Ordenador portátil y conexión Internet	Herramienta para desarrollar el sistema, redactar la documentación y comunicarse.
	Lugar de trabajo	Se incluyen los recursos de la electricidad, el agua y los muebles.
Recurso software	Sistema operativo Ubuntu 16.04 LTS	Sistema operativo que permitirá utilizar las aplicaciones necesarias para el desarrollo del proyecto.
	Google Chrome	Navegador web para el acceso a las herramientas necesarias para el proyecto.
	Atom	Editor de texto para implementar el código.
	Git	Sistema de control de versiones del código.
	Postman	Herramienta para el testing del sistema.

CUADRO 6.1: Tabla de recursos disponibles

6.2.2. Presupuesto

La referencia para las remuneraciones son tomadas del estudio de Michael Page [25], donde un programador y un diseñador reciben como mínimo 18.000 euros/año, un analista, un mínimo de 24.000 euros/año y un jefe de proyecto, un mínimo de 40.000 euros/año. Para estimar los costes de los recursos humanos hay que tomar en cuenta el salario bruto, la Seguridad Social y otros costes del personal (como podría ser los costes de formación). Para eso, una estimación en base a los salarios en bruto de cada uno multiplicado por un factor de 1,35 (tomando en cuenta que el salario bruto suele ser el 75 % aproximadamente del coste total) podemos obtener una aproximación al coste total de cada persona.

Para los costes indirectos, tenemos las siguientes formulas para el calculo de cada recurso:

- *Amortización de portátil*: 1000 euros precio inicial / (8 horas por día * 251 días laborables por año * 4 años de vida) [19]
- *Lugar de trabajo*: 13 euros/m cuadrado * 50m cuadrado oficina = 650 euros/-mes. Total (Seguro 1000 euros/año, muebles, agua, etc): 800 euros/mes = 5 euros/hora
- *Electricidad*: 0.12 kWh portátil * 0.1342 euros/kwh [26]
- *Internet*: 50 euros por mes / (8 horas por día * 20 días por mes)

Actividad	Rol	Coste Estimado (Coste/hora) (Amortización)	Horas Estimadas	Importe
Fase planificación inicial			80h	
Definir objetivos del proyecto	Jefe de proyecto	27€/h	20h	540€
Aprendizaje previo	Programador	13,5€/h	40h	540€
Config. entorno de trabajo	Jefe de proyecto	27€/h	20h	540€
Fase Gestión de proyecto			75h	
Contexto y Alcance	Jefe de proyecto	27€/h	25h	675€
Planificación temporal	Jefe de proyecto	27€/h	8h	216€
Gestión econ. y Sosteni.	Jefe de proyecto	27€/h	9h	243€
Documento completo	Jefe de proyecto	27€/h	18h	486€
Prep. y presentación oral	Jefe de proyecto	27€/h	15h	405€
Fase de implementación			405h	
Definir objetivos de la iteración	Jefe de proyecto	27€/h	30h	810€
Diseñar base de datos	Arquitecto	27€/h	60h	1.620€
Implementar backend	Programador	13,5€/h	75h	750€
Diseñar interfaz de usuario	Diseñador	27€/h	60h	1.012,5€
Implementar frontend	Programador	13,5€/h	90h	1215€
Pruebas de código	Tester	13,5€/h	48h	648€
Documentar resultados	Jefe de proyecto	27€/h	30h	810€
Reunión de control	Jefe de proyecto	27€/h	6h	162€
Reunión de revisión de código	Jefe de proyecto	27€/h	6h	162€
Fase final y cierre			30h	
Redacción memoria y documentación	Jefe de proyecto	27€/h	15h	405€
Preparación Defensa TFG	Jefe de proyecto	27€/h	15h	405€
Total Coste Directo (CD)			**590h**	**11.644,5€**
Ordenador Portátil (Toshiba Satellite P50)		0,12€/h aprox.	590h	70,80€
Lugar de trabajo		5€/h	590h	2950€
Electricidad		0,0161€/h apr.	590h	9,50€
Conexión internet		0,3€/h aprox.	590h	177€
Total Coste Indirecto (CI)				**3207,30€**
Total CD + CI				**14.851,80€**
Contingencia	5% de contingencia			758,80€
Total CD+CI+Contingencia				**15.610,60€**
Iteración adicional	riesgo=40%		80h	604,80€
Corrección de errores	riesgo=20%		20h	54€
Total imprevistos				**658,80€**
TOTAL:				**16.269,40€**

CUADRO 6.2: Tabla de costes

6.2.3. Control de gestión

Para el control de las desviaciones que se puedan producir, tenemos el siguiente sistema.

Para cada tarea, las desviaciones en coste se calcularán con la fórmula (CE - CR)

x CHR, donde CE es coste estimado, CR, coste real, y CHR, consumo de horas real. Y las desviaciones en consumo se obtendrán de (CHE - CHR) x CE, donde CHE es consumo de horas estimado, CHR, consumo de horas real, y CE, coste estimado.

6.3. Sostenibilidad del proyecto

6.3.1. Dimensión económica

El proyecto tiene un presupuesto razonable, dado a las funcionalidades principales que se van a implementar, la aplicación tendrá forma de producto final a la finalización del proyecto. Lo ideal es tener una aplicación que no requiera de mantenimiento a lo largo de su vida útil, pero en la realidad habría que tener a un programador para encargarse de ello. Si queremos plantearnos una manera de sacar beneficio económico de esta aplicación, podemos plantearnos añadir anuncios publicitarios en la aplicación web, recibir donaciones de los usuarios o ofrecer una versión *premium* de la misma con algunas funcionalidades extras o personalizables.

6.3.2. Dimensión ambiental

Para este proyecto, tenemos como puntos positivos a la sostenibilidad ambiental el alojamiento del producto en un servidor de *Heroku* [14] en vez de montar nuestro propio servidor que nos llevaría consumir recursos tanto eléctrico como hardware. Además, *Heroku* [14] pone a dormir nuestra aplicación si en 30 minutos nadie accede y eso ayuda a ahorrar recursos en sus servidores. También tenemos código reutilizable de proyectos anteriores, como la autenticación de usuario o diversos componentes o módulos para integrar en nuestra aplicación. Y el desmantelamiento del producto al final de su vida útil tiene un impacto nulo al medio ambiente, ya que solo tendríamos que retirarlo del servidor.

Como pequeño efecto negativo, tenemos que la aplicación requiere de un ordenador y conexión a internet, lo cual, su consumo tiene cierto impacto ambiental. Para obtener una aproximación en kW de lo que se consumirá durante el proyecto, tenemos las 590 horas de la duración del proyecto, 0,1 kWh de la luz, 0,12 kWh del portátil, y 1kWh del aire acondicionado.

$$590\text{h} \times (0{,}1 + 0{,}12 + 1)\ \text{kWh} = 719{,}8\ \text{kW}$$

6.3.3. Dimensión social

Esta aplicación ofrecerá a los usuarios una manera sencilla y controlada de llevar a cabo una discusión, asistiéndoles a la hora de formular un argumento y organizando el hilo con cierta ergonomía, además de poder tener una biblioteca pública de discusiones y un recopilatorio de tópicos propios. Se cree que con esto se pueda dar soporte a los usuarios para procesar todo tipo de información en un formato de discusión. Todo esto pretende ayudar a las personas a construir una discusión articulada y mejorar la manera en que se piensa gracias a la confrontación de varias ideas.

Personalmente, este proyecto me permitirá desarrollar mis competencias técnicas como la gestión de un proyecto, la documentación del mismo, el diseño de la arquitectura de un software informático y su implementación.

Capítulo 7

Análisis de requisitos

Con los objetivos del proyecto explicados, se pasa a definir los *stakeholders* y los requisitos que tendrá el sistema.

7.1. *Stakeholders*

Para saber los requisitos del sistema, hay que tener en cuenta a los *stakeholders*, es decir, personas interesadas e involucradas en este proyecto. De estos, surgen las necesidades o roles que se deben tomar para que el proyecto cumpla con las objetivos.

A continuación se ven a los involucrados:

7.1.1. Empresa *Prototyp* [1]

La empresa es una consultoría que se especializa en el prototipado de las soluciones tecnológicas que ofrece, mediante su propia metodología *Agile*. Además, desarrollan varios proyectos de todo tipo y tecnologías con el cual consiguen inspirar a posibles clientes para la integración de nuevas tecnologías en sus propias ideas. Se espera que este proyecto sea una muestra más para demostrar la capacidad de las nuevas tecnologías.

7.1.2. Ponente del TFG

El ponente ayudará al autor del TFG en todo tipo de cuestiones para que el proyecto avance adecuadamente y verificará que el conjunto del trabajo se lleve con éxito.

7.1.3. Autor del TFG

Como autor del proyecto, el interés viene de las ganas de aprender y demostrar las capacidades de llevar a cabo la construcción de un nuevo producto, tanto en la parte de gestión y planificación, como en la parte de la implementación. Asumirá los roles de Jefe de proyecto, Analista, Diseñador, Programador y *Tester*.

Como usuario del producto final, hay un interés en el tema de debates y la escritura de ensayos.

7.1.4. Usuario final

Para cualquier usuario, sea escritor o no, con interés en la discusión de ideas o en recopilar diferentes tópicos con argumentaciones propuestas por otros, con el fin de

investigar para sus propios escritos o simplemente el aportar conocimiento con otros usuarios.

7.2. Requisitos funcionales

Los requisitos funcionales para el sistema se recogen en forma de *historias de usuario*, donde se definen unos *criterios de aceptación* que nos permitirá concretar cuando una funcionalidad está finalizada.

#1 Pantalla de discusión	
Descripción	▪ **Como** usuario ▪ **Quiero** ver la pantalla de discusión ▪ **Para** poder leer el título y la descripción de la discusión, las opiniones de cada participante, los argumentos propuestos y los puntos de concordancia y discrepancia.
Criterios de aceptación	▪ Hay que mostrar el título y la descripción. ▪ Hay que mostrar los avatares involucrados con la discusión, con la opinión inicial de cada uno. ▪ Hay que mostrar los argumentos propuestos de cada participante. ▪ Hay que mostrar una tabla con los puntos en los que se esté de acuerdo y los desacuerdos.

CUADRO 7.1: Historia de usuario: *Pantalla de discusión*

#2 Autentificación del usuario mediante Google Sign In	
Descripción	▪ **Como** usuario ▪ **Quiero** identificarme en el sistema ▪ **Para** poder interactuar con sistema.
Criterios de aceptación	▪ Hay que mostrar una ventana emergente para autorizar el acceso del sistema a las credenciales de *Google*. ▪ El usuario debe estar identificado en el sistema hasta que caduque o se cierre la sesión.

CUADRO 7.2: Historia de usuario: *Autentificación del usuario mediante Google Sign In*

#3 Añadir un argumento	
Descripción	▪ **Como** usuario identificado y con, al menos, un avatar asignado ▪ **Quiero** escribir un argumento ▪ **Para** poder proponer el argumento a la discusión.
Criterios de aceptación	▪ Hay que poder escribir el argumento y enviarlo. ▪ Hay que limitar el argumento a 100 palabras como máximo. ▪ Si como usuario tengo asignado los dos avatares, hay que poder escoger uno de ellos para proponer un argumento.

CUADRO 7.3: Historia de usuario: *Añadir un argumento*

#4 Proponer un punto de concordancia o discrepancia	
Descripción	▪ **Como** usuario identificado y con, al menos, un avatar asignado ▪ **Quiero** escribir un punto de concordancia o de discrepancia ▪ **Para** poder proponer el punto de concordancia o de discrepancia a la discusión.
Criterios de aceptación	▪ Hay que poder escribir el punto de concordancia o de discrepancia y enviarlo. ▪ Si como usuario tengo asignado los dos avatares, hay que poder escoger uno de ellos para proponer el punto de concordancia o de discrepancia.

CUADRO 7.4: Historia de usuario: *Proponer un punto de concordancia o discrepancia*

#5 Aceptar o rechazar un punto de concordancia o discrepancia propuesto	
Descripción	▪ **Como** usuario identificado y con, al menos, un avatar asignado ▪ **Quiero** dos botones con opción de aceptar y rechazar en la tabla de puntos de concordancia ▪ **Para** poder aceptar o rechazar el punto de concordancia o de discrepancia propuesto.
Criterios de aceptación	▪ Hay que eliminar los puntos rechazados. ▪ Hay que marcar como aceptados, los puntos aceptados.

CUADRO 7.5: Historia de usuario: *Aceptar o rechazar un punto de concordancia o discrepancia propuesto*

#6 Invitar usuario a participar en una discusión	
Descripción	▪ **Como** usuario identificado y propietario de la discusión ▪ **Quiero** mandar una invitación mediante *email* a un usuario ▪ **Para** invitar al usuario a participar en la discusión.
Criterios de aceptación	▪ El sistema debe mandar la invitación mediante *email*. ▪ El usuario invitado debe identificarse mediante *Google* para participar. ▪ El sistema debe añadir al usuario invitado y identificado a la lista de participantes de dicha discusión.

CUADRO 7.6: Historia de usuario: *Invitar usuario a participar en una discusión*

#7 Asignar participante de la discusión a un avatar	
Descripción	▪ **Como** usuario identificado y propietario de la discusión ▪ **Quiero** asignar a un usuario, participante de la discusión, a un avatar ▪ **Para** que el usuario pueda interactuar en la discusión.
Criterios de aceptación	▪ Hay que permitir al usuario asignado a, al menos, un avatar poder proponer argumentos y puntos de concordancia y discrepancia y aceptar y rechazar dichos puntos con los avatares que tenga asignado.

CUADRO 7.7: Historia de usuario: *Asignar participante de la discusión a un avatar*

#8 Crear nueva discusión	
Descripción	▪ **Como** usuario identificado ▪ **Quiero** crear una nueva discusión ▪ **Para** poder empezar una discusión de un tema en concreto.
Criterios de aceptación	▪ Hay que poder especificar el título y la descripción de la discusión y definir los nombres de los dos avatares con sus respectivas opiniones iniciales.

CUADRO 7.8: Historia de usuario: *Crear nueva discusión*

#9 Borrar una discusión	
Descripción	▪ **Como** usuario identificado ▪ **Quiero** borrar una discusión propia ▪ **Para** eliminar la discusión del sistema.
Criterios de aceptación	▪ Hay que mostrar un cuadro de confirmación para borrar la discusión.

CUADRO 7.9: Historia de usuario: *Borrar una discusión*

#10 Lista de discusiones públicas	
Descripción	▪ **Como** usuario ▪ **Quiero** ver la pantalla del repertorio de discusiones públicas ▪ **Para** explorar y navegar entre las listas de discusiones públicas.
Criterios de aceptación	▪ Hay que mostrar un listado de discusiones públicas. ▪ La lista ha de estar paginada. ▪ Se debe de poder crear y borrar discusiones desde esta pantalla.

CUADRO 7.10: Historia de usuario: *Lista de discusiones públicas*

#11 Lista de discusiones propias	
Descripción	▪ **Como** usuario identificado ▪ **Quiero** ver la pantalla del repertorio de discusiones propias ▪ **Para** navegar entre las listas de discusiones propias.
Criterios de aceptación	▪ Hay que mostrar un listado de discusiones propias. ▪ La lista ha de estar paginada. ▪ Se debe de poder crear y borrar discusiones desde esta pantalla.

CUADRO 7.11: Historia de usuario: *Lista de discusiones propias*

#12 *Fork* de una discusión	
Descripción	▪ **Como** usuario identificado ▪ **Quiero** hacer *fork* a una discusión cualquiera ▪ **Para** poder continuarla como discusión propia.
Criterios de aceptación	▪ Hay que crear una copia idéntica de la discusión. ▪ El propietario de la discusión pasará a ser el usuario identificado. ▪ Los avatares de la discusión estarán asignados al propietario.

CUADRO 7.12: Historia de usuario: *Fork de una discusión*

#13 Discusión privada	
Descripción	▪ **Como** usuario identificado ▪ **Quiero** poner una discusión en privado ▪ **Para** tener un acceso a la discusión restringido al propietario y los participantes.
Criterios de aceptación	▪ Hay que mostrarse en el listado de discusiones propias. ▪ La discusión no debe aparecer en la lista de discusiones públicas.

CUADRO 7.13: Historia de usuario: *Discusión privada*

7.3. Requisitos no funcionales

De los requisitos no funcionales, se toman como plantilla las descritas por *Volere* [27]. Estos requisitos describen cómo han de comportarse las funcionalidades del sistema.

***Volere*: 10a Requisito de apariencia**	
Descripción	Diseño de la interfaz de usuario que facilite el ambiente para una discusión.
Justificación	Con una apariencia que no se ajusta al contexto de una discusión, causa molestias al no permitir al usuario meterse en el ambiente para una discusión.

CUADRO 7.14: Requisito no funcional: *10a Requisito de apariencia*

***Volere*: 10b Requisito de estilo**	
Descripción	Diseño de los estilos adaptado a la actualidad, utilizando estilos similares a *Github* o *Amazon*.
Justificación	Un diseño anticuado desprestigiará la aplicación, por no adaptarse a los nuevos estándares.

CUADRO 7.15: Requisito no funcional: *10b Requisito de estilo*

***Volere*: 11a Requisito de facilidad de uso**	
Descripción	Interfaz de usuario fácil de usar, siguiendo los estándares actuales para la interacción con las funcionalidades.
Justificación	Si se dificulta la interacción con la aplicación, este se vuelve inservible.

CUADRO 7.16: Requisito no funcional: *11a Requisito de facilidad de uso*

***Volere*: 14c Requisito de adaptabilidad**	
Descripción	La aplicación web tendrá que ejecutarse correctamente en los navegadores web de cualquier sistema operativo y la apariencia deberá adaptarse al tamaño, ya sea un monitor de 20 pulgadas o un dispositivo móvil de 3 pulgadas.
Justificación	Para facilitar la accesibilidad a la aplicación.

CUADRO 7.17: Requisito no funcional: *14c Requisito de adaptabilidad*

***Volere*: 15a Requisito de accesibilidad**	
Descripción	Solo los usuarios identificados podrán interactuar con las funcionalidades que necesiten autorización.
Justificación	Para poder asociar las interacciones con los usuarios.

CUADRO 7.18: Requisito no funcional: *15a Requisito de accesibilidad*

Capítulo 8

Especificación

8.1. Esquema conceptual de los datos

Teniendo en cuenta los requisitos definidos en el capítulo anterior, se procede a representar en un diagrama los datos necesarios para cumplir con dichos objetivos.

Primero se expone la descripción de cada modelo propuesto en el diagrama y luego el diagrama en sí.

8.1.1. Descripción de los modelos

- **User**: Representa una persona identificada en el sistema. Puede ser propietario y participar en varias discusiones (*Discussions*) y adoptar el rol de varios avatares (*Avatars*).
- **Discussion**: Representa una discusión. Tiene un título y una descripción del tópico, dos avatares (*Avatars*) y pertenece a un propietario (*User*). Puede tener varios argumentos (*Arguments*), puntos de concordancia (*Agreements*) y participantes (*Participants*).
- **Avatar**: Representa un personaje ficticio que intentará defender una opinión discrepante con su par. Tiene un nombre, una opinión inicial y un usuario (*User*) asignado. Puede tener argumentos (*Arguments*) y puntos de concordancia (*Agreements*) propuestos por el mismo.
- **Argument**: Representa un argumento propuesto en la discusión. Tiene un número de orden, el contenido del argumento y la fecha de publicación. Pertenece a una discusión (*Discussions*) y a un avatar (*Avatars*).
- **Agreement**: Representa un punto de concordancia o discrepancia propuesto en la discusión. Tiene un contenido, un indicador de si es un punto de concordancia o uno de discrepancia y si se ha aceptado la propuesta. Pertenece a una discusión (*Discussions*) y a un avatar (*Avatars*).
- **Comments**: Representa un comentario sobre la discusión. Contiene el texto del comentario Pertenece a una discusión (*Discussions*) y a un usuario (*User*).
- **Criteria**: Representa un criterio para la discusión. Contiene el texto del criterio. Pertenece a una discusión (*Discussions*).
- **Rating**: Representa una valoración sobre un criterio de una discusión. Contiene la valoración. Pertenece a un avatar (*Avatar*) y a un criterio (*Criteria*).

8.1.2. Diagrama de datos

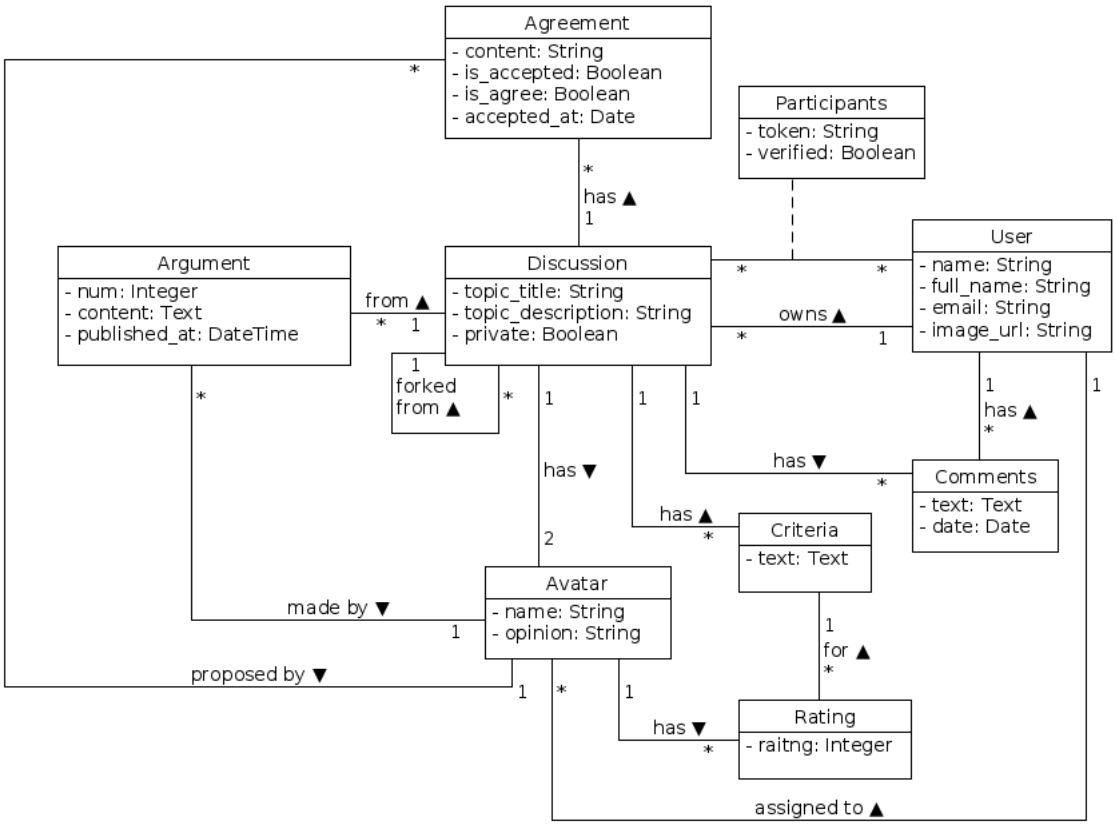


FIGURA 8.1: Diagrama de datos

Restricciones textuales:

1. Claves externas: (*Participants*, token);
2. El propietario (*User*) de una discusión (*Discussion*) también debe ser participante de la discusión (*Discussion*).
3. Un avatar (*Avatar*) solo puede estar asignado a un usuario (*User*) que participe en la discusión (*Discussion*).
4. Una valoración (*Rating*) debe pertenecer a un criterio (*Criteria*) y un avatar (*Avatar*) de la misma discusión (*Discussion*).

Capítulo 9

Diseño

Una vez se tiene la especificación del sistema a construir, se pasa a la fase de diseño, donde habrá que decidir qué infraestructura tendrá el sistema, que arquitectura software y patrones se aplicarán y como se diseñará la base de datos.

9.1. Infraestructura

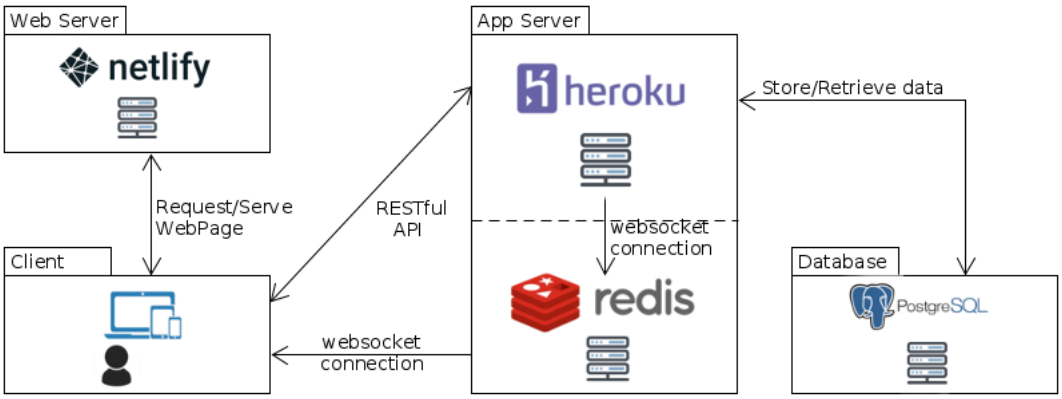


FIGURA 9.1: Diagrama de infraestructura

El sistema consistirá en una plataforma web, separada en una parte *Frontend* y otra de *Backend*. La parte *Frontend* se alojará en el servidor de *Netlify*, donde se servirá la aplicación con las diferentes vistas o pantallas. La parte *Backend* se alojará en el servidor de *Heroku*, que se encargará de resolver las llamadas *API* que se manda por *Frontend*. Desde *Heroku*, el *business logic* del *Backend* se comunicará con la base de datos de *PostgreSQL*. Para lograr la conexión *WebSocket* con el *Frontend* y el *Backend*, se hará mediante *Redis*, que permitirá mandar datos a tiempo real.

¿Por qué un *Frontend* y un *Backend* separados? En primer lugar, porque la tecnología actual facilita la construcción de estas aplicaciones por separado hasta tal punto que, en algunos casos, es incluso más fácil hacerlo de esta manera. Tener una parte del sistema que gestiona la lógica independientemente del resto de la aplicación y comunicarse mediante una *API*, puede ser luego reutilizada por otras aplicaciones. En *Heroku* solo tendrá que ocuparse con la parte servidor o *Backend*, mientras *Netlify* se encarga de repartir la parte del cliente. También, al tener un *Frontend* y un *Backend* desacoplados, permite un desarrollo de ambos con independencia del uno u otro, esto se traduce a poder trabajar en una parte sin preocuparse de estropear la otra e incluso poder trabajar las dos simultáneamente.

¿Por qué *Netlify*? Queriendo minimizar los costes de alojamiento, hay dos opciones de *PaaS*[1] para poder servir la parte de cliente de la aplicación web, *Heroku* y *Netlify*. *Netlify* solo permite alojar páginas web estáticas, a diferencia de *Heroku*, el cual ofrece también alojar la parte servidor. Los dos tienen un plan gratuito, pero la desventaja de *Heroku* es el poner a dormir la aplicación a los 30 minutos de inactividad. Como solo hay que servir la parte cliente de la web, se ha cogido *Netlify* como opción.

¿Por qué *Heroku*? *Heroku* es un *PaaS* que ofrece un plan gratuito para alojar aplicaciones web, con opción de migrar a otros planes según los recursos que se vayan a necesitar en el futuro, el cual permite una escalabilidad muy fácil de dichas aplicaciones.

¿Por qué *Redis*? *Redis* es un sistema de almacenaje de estructuras de datos en memoria y de código abierto. Dicho sistema también viene integrado a *Heroku* como *Add-on*, y permite montar la arquitectura para las conexiones de *WebSockets*.

¿Por qué *PostgreSQL*? *PostgreSQL* es un sistema de gestión de bases de datos relacional y de código abierto. Dicho sistema también viene integrado a *Heroku* como *Add-on*, con la ventaja de tenerlo todo configurado por defecto para que, cuando se despliegue el código de una aplicación, funcione sin tener que molestarse en configurar la conexión a la base de datos.

9.2. Arquitectura software

Visto la infraestructura física, ahora se procede a presentar la arquitectura que se ha decidido tener para la aplicación de este proyecto.

El sistema está dividido en 3 capas. La capa de presentación o *View Layer*, la capa de dominio o *Business Logic Layer* y la capa de persistencia o *Persistence Layer*. La capa de presentación se encarga de presentar los resultados y recibir los eventos y peticiones del usuario. La capa de dominio, satisface las peticiones del usuario, pero ignora el cómo presentarlos y cómo guardar los datos. La capa de persistencia, sabe como guardar los datos y cómo están estructurados, pero ignora el cómo tratarlos. De esta manera, se puede independizar las tres partes del sistema para poder facilitar la comprensión del código en general.

En la capa de presentación, se implementa las pantallas para interactuar con el usuario. Consume las *RESTful API* (más, en el siguiente apartado) para comunicarse con la capa de dominio.

En la capa de dominio, se implementa las *RESTful API*, con funciones modulares para ofrecer la lógica del sistema a la capa de presentación. También se comunica con la capa de persistencia.

Y en la capa de persistencia, se estructura los datos para ser alojados en la base de datos. Se utiliza un sistema de gestión de base de datos para gestionar las estructuras de datos y los datos en sí.

[1] Platform as a Service (Plataforma como servicio).

9.3. Patrón *RESTful API*

Una *RESTful API* es un patrón de arquitectura del software donde el objetivo que tiene es crear una interfaz entre dos conexiones HTTP para la transferencia de datos.

El patrón consiste en dividir una transacción en pequeños módulos, los cuales se encargan de una parte de la transacción. Esto ofrece a los desarrolladores más flexibilidad comparada con una transacción entera. También tiene como característica el no mantener ningún estado (*stateless*) ni en el cliente ni en el servidor, es decir, cada petición contiene toda la información necesaria para su ejecución.

Toda la documentación de las peticiones *RESTful API* están en el apéndice A.

9.4. Base de datos

En cuanto a la base de datos, se ha traducido el esquema conceptual de los datos de la especificación a una base de datos relacional. En el siguiente diagrama se definen todos los atributos que tendrán las tablas de la base de datos.

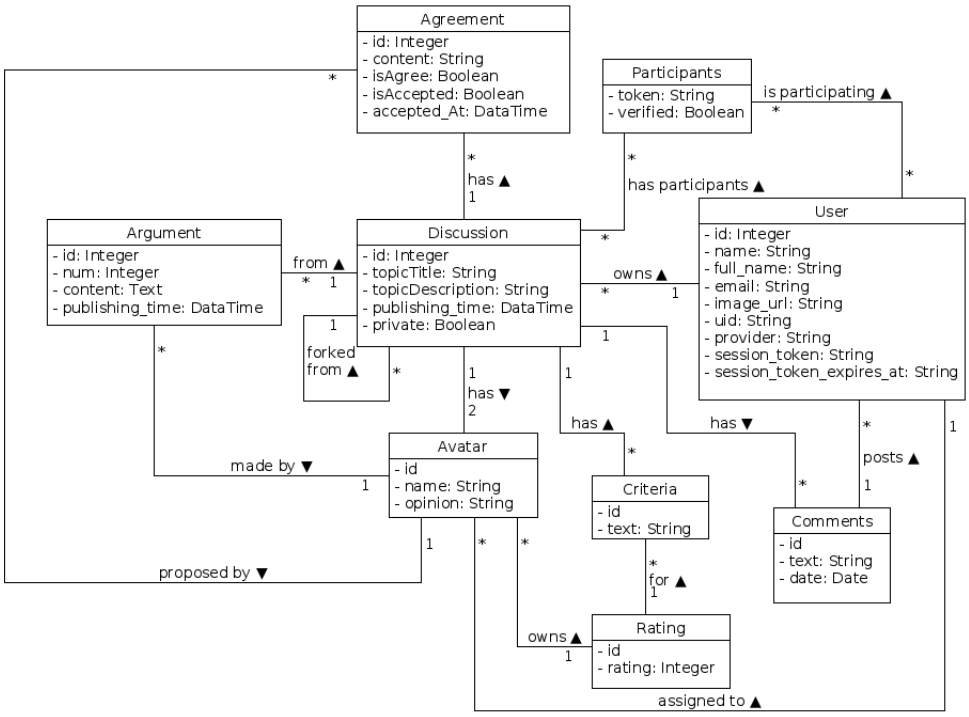


FIGURA 9.2: Diagrama de clases

9.5. Autenticación del usuario

Para la autenticación del usuario, se ha seguido el siguiente proceso acorde con el patrón propuesto por *Auth0* [28], con el fin de identificar al usuario en el sistema.

Como vemos en la siguiente figura, el proceso sigue estos pasos:

1. El usuario pide identificarse mediante un botón desde *Frontend*.

2. El servicio pide la información del usuario, como el *email* y contraseña de la cuenta.
3. El usuario envía la información.
4. El servicio pide la autorización para el acceso del sistema a los datos del usuario.
5. El usuario acepta la autorización.
6. El servicio envía el *idToken* generado para el usuario al *Frontend*.
7. El *Frontend* envía el *idToken* al *Backend*.
8. El *Backend* verifica el *idToken* y genera un *sessionToken* con un tiempo de caducidad, que finalmente se envía al *Frontend*.

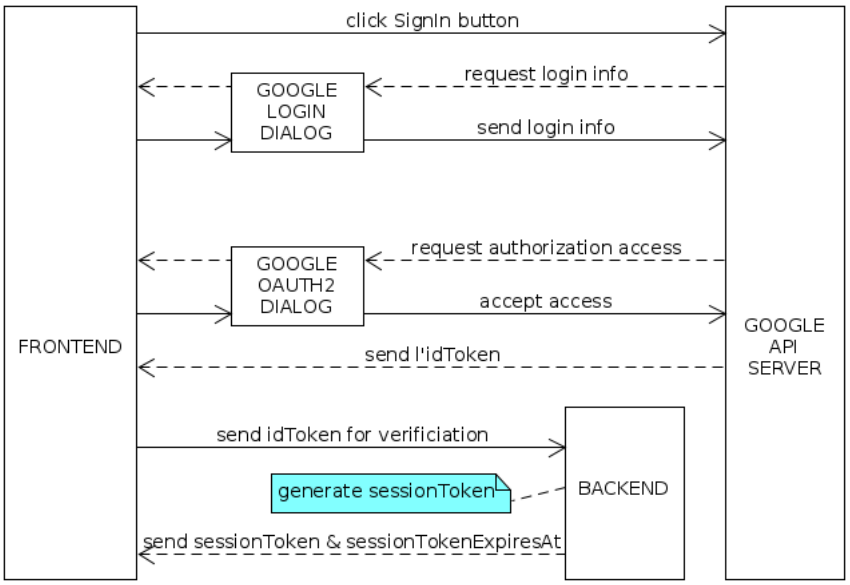


FIGURA 9.3: Proceso de autenticación del usuario

9.6. Tiempo real

Para la funcionalidad del sistema en tiempo real, se ha seguido el siguiente patrón para poderse integrar con la arquitectura base, explicada en el apartado 9.2, y reducir el máximo posible la redundancia en el código para la lógica del sistema.

9.6.1. Patrón *Publisher-Subscriber* [29]

Este patrón consiste en dos tipos de componentes, un *Publisher* y un *Subscriber*. Los que envían mensajes, los llamados *Publishers*, desconocen al receptor, los *Subscribers*, y lo que se hace es almacenar estos mensajes en un *channel* y los *Subscribers* se encargan de escuchar dicho *channel* y recibir estos mensajes. Puede haber varios *channels* para poder enviar diferentes tipos de información y los *Subscribers* pueden suscribirse a cualquiera que les interese.

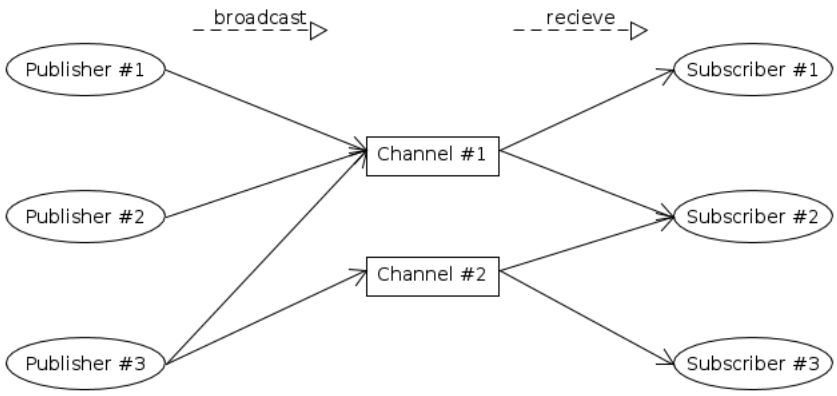


FIGURA 9.4: Proceso de autenticación del usuario

9.6.2. Integración del patrón *Publisher-Subscriber* [29]

Al integrar este patrón a la arquitectura base, quedaría la siguiente estructura.

Desde el *Frontend*:

- Se hace una petición HTTP GET para rellenar la vista con la información que hay hasta ahora.
- Se crea una nueva conexión, a través del *WebSocket*, y se suscribe a los evento específicos que necesite la vista.
- A medida que se reciban nuevos eventos, la vista irá actualizando la información pertinente.

Desde el *Backend*:

- Recibirá peticiones HTTP y las gestionará.
- Cuando haya que notificar un evento, se publicará a través del *WebSocket*, enviando en el contenido del mensaje la información necesaria.

Capítulo 10

Implementación

Una vez se tiene todos los elementos de la parte de diseño definidos, se procederá a explicar qué decisiones se han tomado para cumplir con los objetivos. A continuación, se lista brevemente, como recordatorio, los objetivos propuestos en el capítulo anterior:

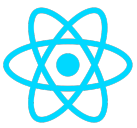
- Conseguir una separación entre capa de presentación, para mostrar las vistas que interactúan con el usuario, y capa de dominio, para servir la *RESTful API*.
- Conseguir una manera de interactuar entre la capa de dominio y la de persistencia.
- Poder enviar *emails* automáticos.
- Conseguir la conexión *WebSocket*.
- Autentificar al usuario.

En los siguientes apartados se expone la tecnología implicada y la metodología seguida.

10.1. Tecnologías implicadas

10.1.1. *Framework* y librerías *Frontend*

Para la implementación de la parte *Frontend*, se ha cogido dos librerías de código abierto, *React JS* y *Semantic UI*, que ofrecen mucha facilidad y flexibilidad a la hora de construir software:



React JS [30]

Es una librería de *JavaScript* [31] enfocada a la construcción de interfaces de usuario. Estas interfaces se construyen a partir de componentes. Un componente contiene su propio código de funcionalidad y de presentación encapsulado, y a su vez, se puede basar de otros componentes para lograr funciones más complejas. Esto permite la reutilización de código y, como resultado, existe un vasto repositorio de componentes compartidos por otros desarrolladores con el objetivo de hacer el desarrollo de una aplicación web una tarea fácil.

React Semantic UI [32]

La librería ofrece un paquete completo de componentes para la presentación de la interfaz de usuario, de un aspecto moderno y profesional. Dicha librería es de mucha ayuda para usuarios sin experiencia en diseño dado que los estilos vienen ya hechos.

A continuación, se ve en la siguiente figura, como está estructurado la interfaz de la aplicación:

- Cada rectángulo rojo representa un componente.
- Cada componente puede utilizar otros componentes.
- Un componente es reutilizado varias veces.

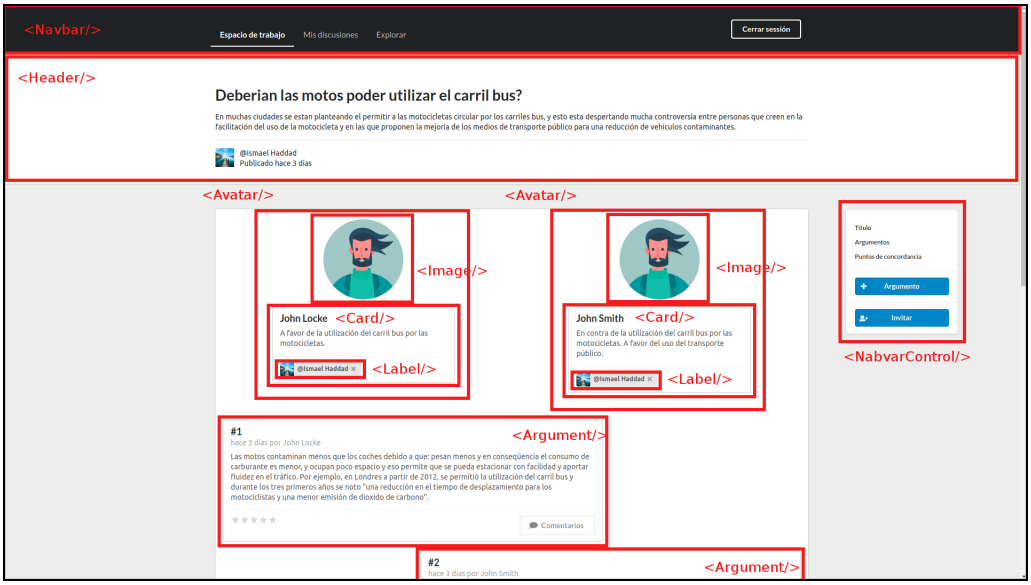


FIGURA 10.1: Captura de pantalla de la interfaz de usuario de la aplicación con indicaciones del esquema de componentes

10.1.2. *Framework Backend*

Ruby on Rails [33]

Es un *framework* de aplicaciones web en *Ruby* [34], basado en el patrón MVC. Ofrece varios comandos para generar una API como, por ejemplo, funciones para el controlador, vistas en formato JSON y migraciones para la base de datos. Utiliza un potente ORM que permite mapear los modelos a la base de datos sin ningún esfuerzo, así mismo, se consigue una comunicación con la capa de dominio y la de persistencia fácilmente. También incorpora *ActionCable* [35], un módulo para montar conexiones *WebSocket* [6], que permite enviar y recibir mensajes mediante canales.

10.1.3. Servicios externos

Google Sign-In [36]

Este servicio permite autentificar a los usuarios mediante sus cuentas de *Google*, como las que usan para *Gmail*, *Youtube*, *Drive* y muchos más. Ofrece una manera de identificar al usuario rápidamente sin que tenga que registrarse a través de formularios y demás.

SendGrid [37]

Esta plataforma ofrece un servicio para mandar *emails* automáticos que permitirá enviar las invitaciones para participar en las discusiones.

10.1.4. Herramientas

Pivotal Tracker [10]

Esta herramienta ofrece diversas tablas para planificar un proyecto de naturaleza *Agile*. Crea un *backlog* a partir de las historias de usuario que se añaden. Dichas historias pueden ser monitorizadas a través de información como el estado, descripciones de detalles particulares de cada uno y los puntos de esfuerzo asignados.

Git [16]

Es un sistema de control de versiones, es decir, permite guardar diferentes versiones del código a través del tiempo. Además, se pueden crear diferentes ramas para trabajar en diferentes versiones de código simultáneamente y poder juntar al final dicha bifurcación.

Github [8]

Github es una plataforma que permite alojar repositorios de código versionado con *git*. y ofrece la posibilidad de colaborar con otras personas mediante los *Pull Request*.

Travis CI [13]

Es un servicio de integración continua que se utiliza para ejecutar las pruebas y desplegar proyectos de software alojados en *Github* automáticamente.

10.2. Proceso seguido

Con respecto a la metodología, se sigue un claro procedimiento con la ayuda de las herramientas mencionadas en el apartado anterior. Dicho esto, se expone a continuación, qué pasos se han seguido.

Con *Pivotal Tracker*, se han definido las *features*, en formato de *historias de usuario*, según los requisitos existentes y se han agrupado en grupos, llamados *Epics*, según la relación entre ellas. En las siguientes figuras, se ven el *Backlog* y las *Epics*, respectivamente.

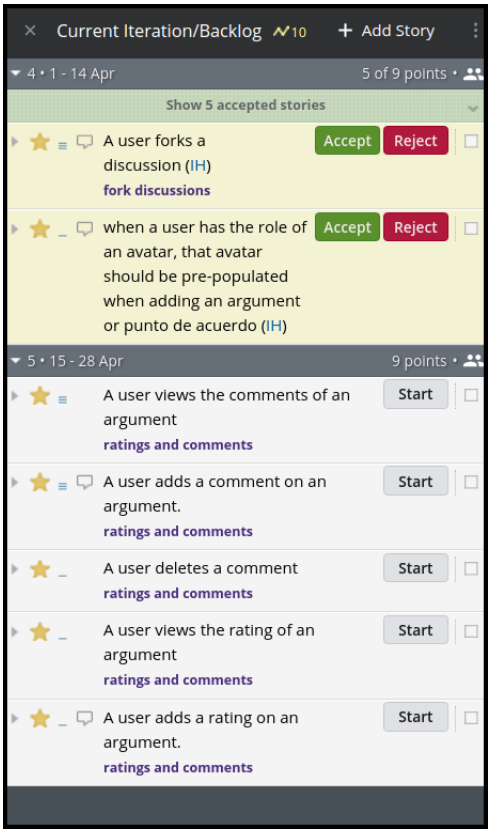


FIGURA 10.2: Captura de pantalla del Backlog de Pivotal Tracker

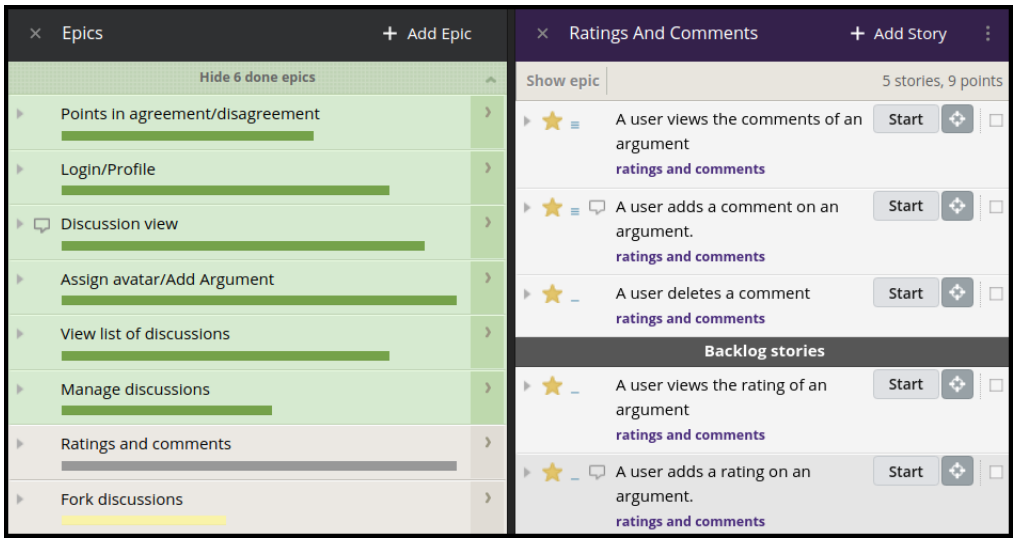


FIGURA 10.3: Captura de pantalla de las Epics de Pivotal Tracker

Después de asignar puntos de esfuerzo para cada *feature* y ordenarlas por prioridad, se da por completada la tarea de planificación y se prosigue a la implementación.

En el siguiente paso, se coge la primera *feature* del *Backlog* y se procede a diseñar dicha funcionalidad. En este caso, se ha empezado con *mock ups* de las pantallas involucradas y las interacciones necesarias a implementar. Luego, para la parte *Backend*, se diseña la base de datos, si hay que añadir, modificar o borrar algún atributo o tabla, y las funciones del controlador necesarias.

A la hora de escribir el código, se sigue el siguiente esquema, para organizar el flujo de trabajo y el versionado del código, según el método de Vincent Driessen [38].

Consiste en 5 tipos de *branches*, o ramas (véase la figura 10.4):

- *Master*. Aquí va el código final que se despliega a producción.
- *Develop*. El código de las funcionalidades ya desarrolladas.
- *Feature*. Las funcionalidades en desarrollo.
- *Release*. Los retoques finales antes de hacer *merge* con *Master*.
- *Hotfix*. Para arreglos críticos, que demandan respuestas rápidas.

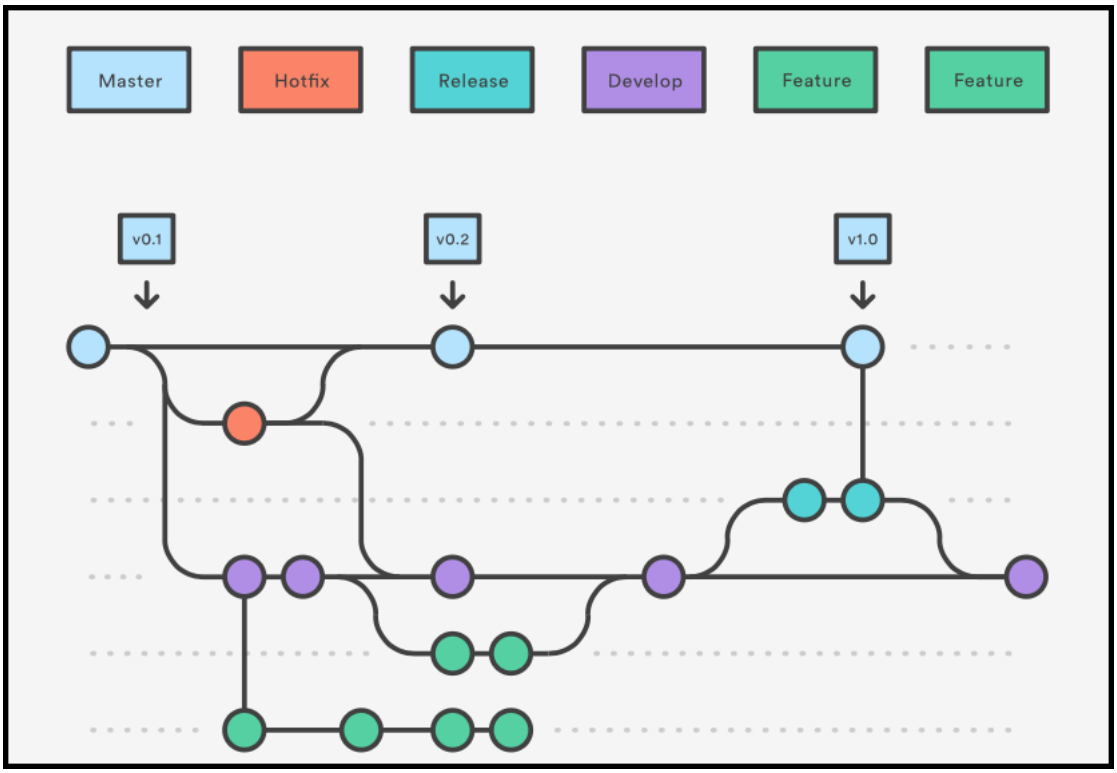


FIGURA 10.4: Diagrama de flujo del método Git Flow [39]

Dicho esto y partiendo de los diseños hechos anteriormente, se procede a crear una nueva *branch* de tipo *feature*. Al acabar de escribir el código, se hace un *Pull Request*, para poder revisar los nuevos cambios del código antes de hacer un *merge* con la *branch Develop*.

Para poder seguir el rastro de las *features* listadas en *Pivotal Tracker* con sus respectivos *Pull Requests*, se usa la función de *tracking*, que añade dicho *Pull* en una sección de cada *historia de usuario* como vemos en la figura 10.6.

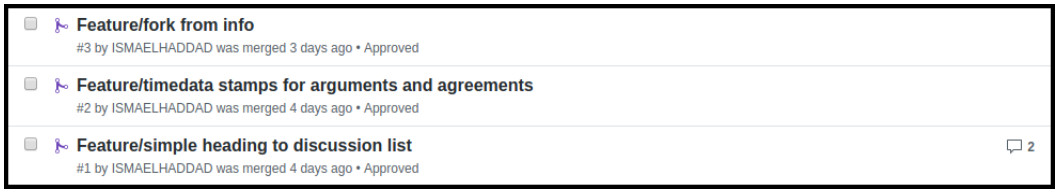


FIGURA 10.5: Ejemplo de lista de Pull Requests en Github

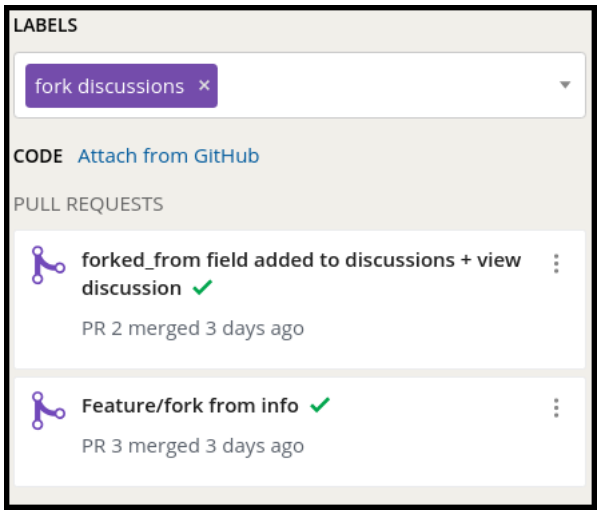


FIGURA 10.6: Ejemplo de tracking de Pull Requests en Pivotal Tracker

Una vez finalizada la implementación de las *features* correspondientes a la iteración actual, se hace un lanzamiento de una *release*. En dichas *releases*, se les asignan unas etiquetas que indican la versión del estado de la aplicación. La convención para las etiquetas, ha sido la siguiente:

[iteration-X.Y.Z] donde:

- X es el número de iteración en la que se hace la *release*.
- Y, el número de versión de la aplicación, que se incrementa cada vez que se añaden nuevas *features*.
- Z, el número de versión de la aplicación, que se incrementa cada vez que hay un nuevo *hotfix*.

Una vez los cambios hayan subido a *Github*, *Travis CI* automáticamente acciona un seguido de ejecuciones que consisten en probar y desplegar el código a producción. Si una de las dos acaba en error, *Travis CI* se encarga de mandar un mensaje con dicho aviso.

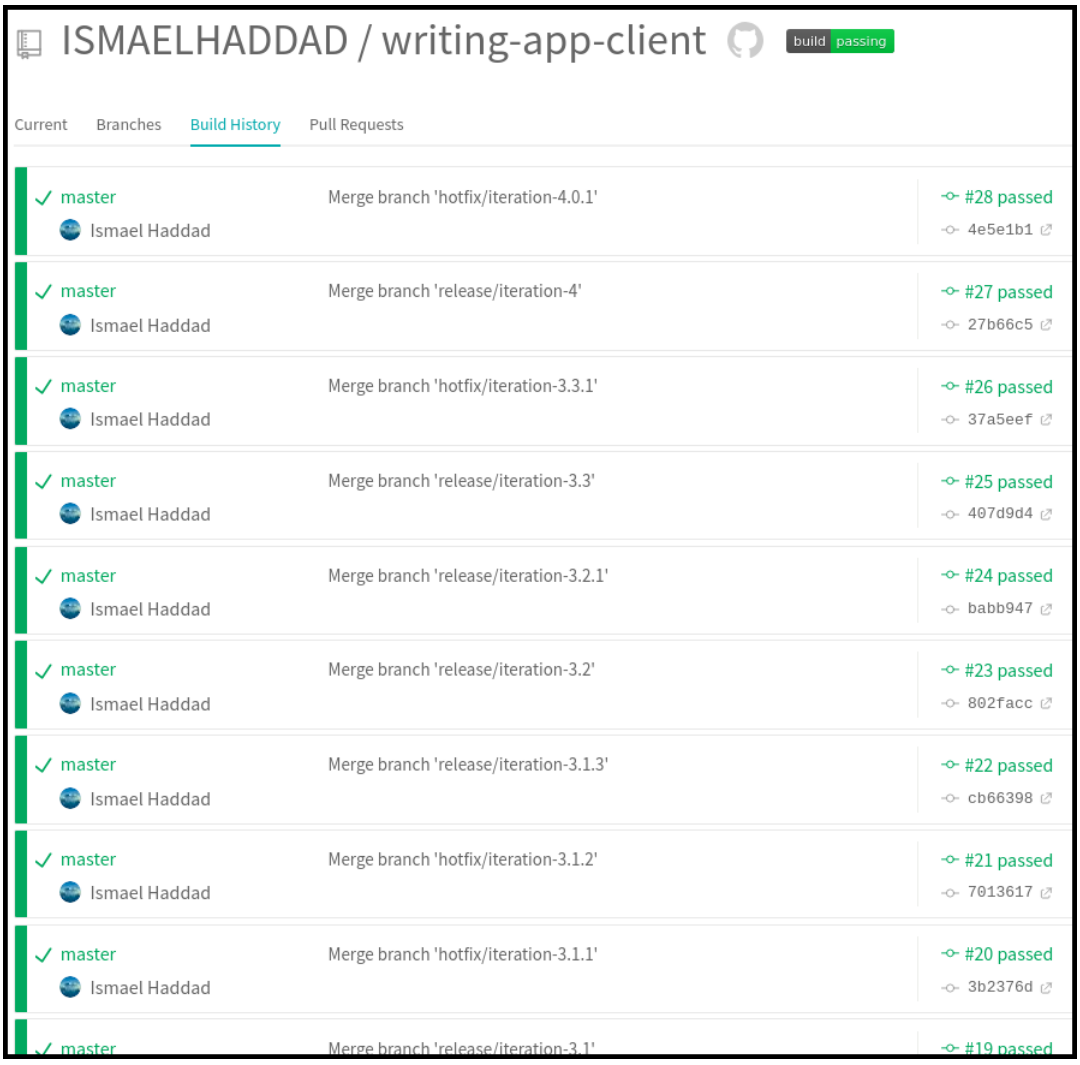


FIGURA 10.7: Captura de pantalla de las ejecuciones de Travis CI

10.3. Estructura del código

Los códigos están subidos en los siguientes repositorios de *Github*:

- (*Frontend*) `https://github.com/ISMAELHADDAD/writing-app-client`
- (*Backend*) `https://github.com/ISMAELHADDAD/writing-app-backend-api`

A continuación se describe como esta estructurado el código de los dos lados del sistema.

10.3.1. *Frontend: React js*

La filosofía de *React js* consiste en construir una página web a base de composiciones de componentes. Estos componentes tienen que encapsular su propia lógica de manera que se puedan utilizar en conjunto con otros componentes, incluso con componentes escritos por otros usuarios.

En el caso de este proyecto, se ha diferenciado un tipo de componentes y se les ha

llamado *containers*. Estos, son los diferentes contenedores para las diversas páginas. Podemos ver la estructura de las carpetas en la figura 10.8.

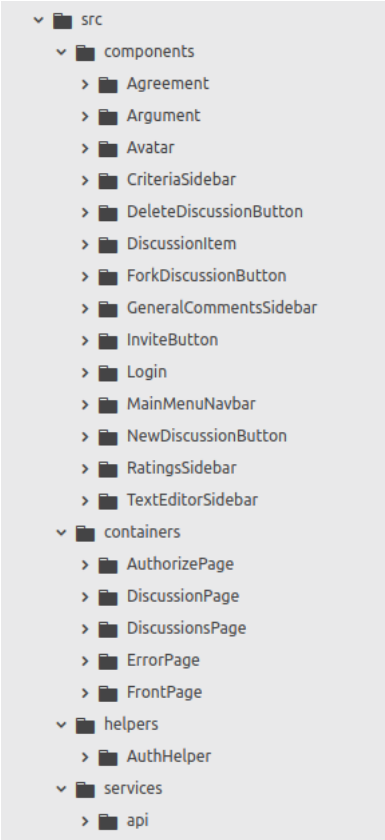


FIGURA 10.8: Estructura del código (*React js*)

Las diferentes páginas se accederán a las rutas definidas en el archivo App.js. (Fig. 10.9)

```
<Route exact path="/authorize"
<Route exact path="/error"
<Route path="/discussion/:id"
<Route path="/my-discussions"
<Route path="/explore"
```

FIGURA 10.9: Rutas a las diferentes páginas

Para ver como están formados los componentes, se toma como ejemplo el componente de la página de discusión: *<DiscussionPage/>*

Como se observa en la figura 10.10, en el constructor se definen los datos que contendrá el componente (llamado *state*) y se inicializarán los *sockets*.

Una vez el componente este montado (es decir, instanciado en el código), se llama a la función *componentDidMount* donde se hará la petición REST API para obtener los datos de la discusión del *Backend* y popular las vistas.

Y cada vez que haya una actualización en el sistema, se recibirán los datos mediante la conexión *sockets* y se procesarán con la función *processDataReceivedFromSocket*.

```
class DiscussionPage extends Component {

  constructor(props) {
    super(props);
    this.state = {
      discussion: {}
    };
    this.createSocket()
  }

  processDataReceivedFromSocket = (data) => {
    if (data.type === 'argument') {…}
    else if (data.type === 'agreement-propose') {…}
    else if (data.type === 'agreement-accept') {…}
    else if (data.type === 'agreement-reject') {…}
    else if (data.type === 'participant-verified') {…}
    else if (data.type === 'avatar-assign') {…}
  }

  componentDidMount() {
    API.getDiscussion(this.props.match.params.id)
      .then(discussion => {
        this.setState({
          discussion: discussion
        })
      })
  }
```

FIGURA 10.10: Inicialización del componente *DiscussionPage*

Para construir el *html* se utiliza la función *render*, donde se declaran los diversos elementos y componentes en un formato de etiquetas (Fig. 10.11).

```
render() {
  return (
    <div>
      {/* Data stored on state of the component is passed as props to child component*/}
      <Avatar avatarData={this.state.discussion.avatar}/>

      {/* Iterate for each argument of the discussion and pass the data as props to child component*/}
      {this.state.discussion.arguments.map((item) => (
          <Argument key={item.num} argumentData={item}/>
      ))}
    </div>
  );
}
```

FIGURA 10.11: Renderización del componente *DiscussionPage*

10.3.2. *Backend: Ruby on Rails*

Ruby on Rails viene con una estructura de archivos ya definido con la justificación de facilitar el desarrollo de aplicaciones sin tener que configurar de más. En la siguiente figura 10.12 vemos dicha estructura.

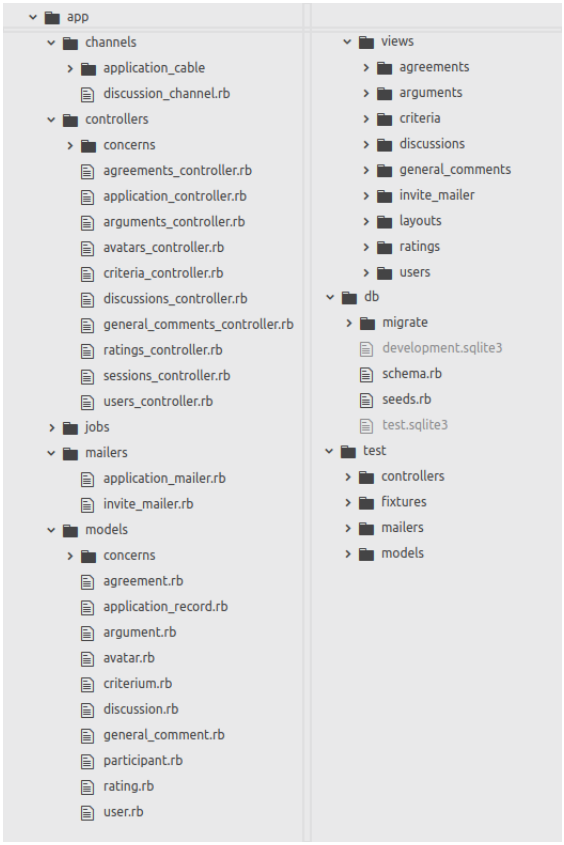


FIGURA 10.12: Estructura del código (*Ruby on Rails*)

Los controladores contienen las funciones para realizar las diversas transacciones de la capa de dominio. En la siguiente figura (10.13) se muestra la acción de crear un nuevo argumento.

```ruby
def create

  @argument = Argument.new(
    num: (Discussion.find(add_argument_params[:discussion_id].to_i).arguments.size + 1),
    content: add_argument_params[:content],
    avatar_id: add_argument_params[:avatar_id].to_i,
    discussion_id: add_argument_params[:discussion_id].to_i,
    publish_time: DateTime.now
  ) if Avatar.find(add_argument_params[:avatar_id]).user.id == current_user.id # Verify if avatar is assigned to the user

  if @argument && @argument.save

    # Emit that new argument has been published
    content = {…}
    ActionCable.server.broadcast 'discussion_room_#' + @argument.discussion.id.to_s,
      type: 'argument',
      content: content.to_json

    # Render the new argument
    render :show, status: :created, resource: @argument

  else

    render json: { message: 'Invalid data'}, status: :unprocessable_entity

  end

end
```

FIGURA 10.13: Crear un argumento (controlador *Ruby on Rails*)

Se crea una instancia de *Argument* con los parámetros correspondientes y se persiste en la base de datos. Si la persistencia es exitosa, se emite el nuevo argumento por *websocket* y, al mismo tiempo, responde la petición mediante HTTP.

También se observar el ejemplo de la invitación mediante email (Fig. 10.14). Se crea un *Participant* sin verificar y se envía un email con el token para la verificación.

```
def invite

  # Only owner of the discussion can make invitation
  if @discussion.user.id != current_user.id
    render :json => {message: "Only owner of the discussion can make invitation"},
      status: :unauthorized
    return
  end

  participant = Participant.new(discussion: @discussion, verified: false)
  Participant.generateUniqueToken(participant)

  if participant.save

    InviteMailer.invite_participant(
      invite_params[:email],  # Email to
      current_user.name, # Username who makes the invite
      @discussion.id, # Discussion involved
      participant.token # Token for identifying the invitation
    ).deliver_now if Rails.env.production?

    render :json => {message: "Invitation send"}, status: :ok

  else

    render :json => {message: "Couldn't generate an invitation"}, status: :unprocessable_entity

  end

end
```

FIGURA 10.14: Invitar a participar (controlador *Ruby on Rails*)

Finalmente, la base de datos se define en el fichero db/schema.rb. Dicho fichero es una abstracción que permite cambiar de bases de datos con facilidad con solo un comando (Fig. 10.15).

```ruby
ActiveRecord::Schema.define(version: 2019_04_29_101617) do

  create_table "users", force: :cascade do |t|
    t.string "name"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.string "full_name"
    t.string "email"
    t.string "uid"
    t.string "provider"
    t.string "session_token"
    t.datetime "session_token_expires_at"
    t.string "image_url"
  end

  create_table "discussions", force: :cascade do |t|
    t.string "topic_title"
    t.string "topic_description"
    t.integer "user_id"
    t.datetime "created_at", null: false
    t.datetime "updated_at", null: false
    t.integer "arguments_count"
    t.boolean "private"
    t.integer "forked_from"
    t.index ["user_id"], name: "index_discussions_on_user_id"
  end
```

FIGURA 10.15: Esquema de base de datos (*Ruby on Rails*)

Capítulo 11

Pruebas de calidad

Para asegurar que la aplicación funcione como es debido y evitar que surjan errores sorpresa, en Ingeniería de Software, se usan como método los llamados *Tests*. Estos consisten en probar que el código se comporte como lo esperado. Las pruebas se dividen en manuales y automáticas.

De las automáticas, hay varios tipos de *Tests* y cada uno se diferencia del resto por cubrir cierto rango del código. Para este proyecto, se han utilizado los unitarios y los funcionales, para probar los modelos y las peticiones REST API, respectivamente. Los *Tests* unitarios verifican si los modelos cumplen con sus restricciones, tales como asegurar la presencia de datos en campos obligatorios o prohibir datos de tipos no correspondientes. Los *Tests* funcionales comprueban que las peticiones REST API devuelven los resultados esperados según los datos de entrada. Este tipo de *Test* se encarga de verificar que se cumplen los requisitos especificados para dicha funcionalidad.

Por otro lado, las pruebas manuales implican verificar el funcionamiento de la aplicación con la ayuda de usuarios. Estas pruebas se hacen a nivel de la interfaz de usuario y llamando a las REST API.

A continuación, vemos que proceso se sigue para implementar las pruebas, y la dicha implementación.

11.1. Metodología para las pruebas de calidad

En la parte *Backend*, a la hora de implementar una nueva funcionalidad, se escriben primero los *Tests* mínimos para exigir que se cumplan los requisitos para esta. Estos *Tests* pueden ser unitarios si se añade un nuevo modelo al sistema y siempre habrán *Tests* funcionales para asegurar un buen funcionamiento de la petición REST API.

Una vez se escribe todo el código necesario para que pasen los *Tests*, se añaden más, si es necesario, para cubrir las peculiaridades de la función especifica. Esto se hace porque, en la metodología *Agile* se suele tener un tiempo limitado y se hace una planificación mínima ya que, antes de implementar el código de una funcionalidad, no se sabe los detalles de la misma y no se puede prever los *Tests* que se necesitará para esta.

En cuanto a la parte *Frontend*, las pruebas se hacen manualmente interactuando con la interfaz de usuario. En las reuniones de retrospectiva se aceptan las historias de usuario si todo funciona bien o se rechazan si se encuentran errores. Estos errores

son tolerables tenerlos en la parte *Frontend* ya que no afecta la parte crítica del sistema. Por el contrario, el *Backend* contiene la parte crítica y por eso se cubre todo el código con pruebas automáticas.

11.2. Implementación de los *Tests*

Para implementar los *Tests* automáticos, se ha utilizado la librería *Mini Tests* [17] para el *framework Ruby on Rails* [33]. Esta permite escribir todo tipo de pruebas, de las cuales se han utilizado las unitarias y las funcionales.

Con los *Tests* unitarios, llamados *Model Tests*, se comprueba que las restricciones de los modelos se cumplan. En el siguiente fragmento de código de la figura 11.1, se observa como se hacen aserciones para cada campo del modelo y esto ayuda a la hora de manipular dichos modelos a no estropear una parte de la misma.

```
3  class DiscussionTest < ActiveSupport::TestCase
4    before do
5      @subject = discussions(:discussion_1)
6    end
7
8    test 'is valid' do
9      # Association test
10     must have_many(:arguments).dependent(:destroy)
11     must have_many(:agreements).dependent(:destroy)
12     must have_many(:avatars).dependent(:destroy)
13     must have_many(:participants).dependent(:destroy)
14     must belong_to(:user)
15     # Validation tests
16     must validate_presence_of(:user_id)
17   end
18 end
```

FIGURA 11.1: Código de las pruebas unitarias

Por otro lado, tenemos los *Tests* funcionales, llamados *Controller Tests*, que comprueban que se devuelva los resultados esperados según la entrada que reciba. Estos *Tests* son importantes para asegurar que se cumplen los requisitos definidos en la especificación para la REST API. También es de mucha ayuda a la hora de hacer modificaciones en una función del controlador y evitar estropearla. En la siguiente figura 11.2 se ve un ejemplo de como se definen estos *Tests*.

```ruby
class DiscussionsControllerTest < ActionDispatch::IntegrationTest
  test "should get show" do
    discussion = discussions(:discussion_1)

    get discussion_url(id:1)

    json = JSON.parse(response.body)
    assert_equal discussion.id, json['id']
    assert_equal discussion.topic_title, json['topicTitle']
    assert_equal discussion.topic_description, json['topicDescription']
    assert_equal discussion.user.id, json['owner']['id']
    assert_equal discussion.user.name, json['owner']['name']
    assert_equal discussion.user.image_url, json['owner']['imageUrl']
    assert_equal 1, json['participants'].size

    assert_response :success
  end

  test "should not get show" do
    get discussion_url(id:100)
    assert_response :not_found
  end

  test "should get index" do
    get discussions_url()

    json = JSON.parse(response.body)
    assert_equal 1, json['discussions'].size, "should get 0 discussion"
    assert_equal 1, json['pages']['current'], "should get current pages 1"
    assert_equal 1, json['pages']['total'], "should get total pages 1"

    assert_response :success
  end

  test "should get index on specific user" do
    get discussions_url(user_id: 1)

    json = JSON.parse(response.body)
    assert_equal 1, json['discussions'].size, "should get 1 discussion"
    assert_equal 1, json['pages']['current'], "should get current pages 1"
    assert_equal 1, json['pages']['total'], "should get total pages 1"

    assert_response :success
  end
```

FIGURA 11.2: Código de las pruebas funcionales

11.3. Pruebas manuales

A medida que se ha ido implementando el *Backend* de la aplicación, se ha utilizado *Postman* [24] para ir realizando pruebas manuales y asistirse de ello para desarrollar el código. *Postman* [24] es una herramienta que permite realizar peticiones REST API donde introduces el *endpoint* y los parámetros de entrada, y devuelve el resultado.

Por otro lado, las pruebas del conjunto de la aplicación se han hecho manualmente, comprobando que las funcionalidades cumplían con los requisitos antes de dar por satisfactorio todos los criterios de aceptación para dicha funcionalidad.

Capítulo 12

Manual de usuario

A continuación, se describen las diversas funcionalidades de la aplicación y su correcto uso.

12.1. Pantalla de discusión

En la figura 12.1, se ve la vista general de la pantalla de discusión.

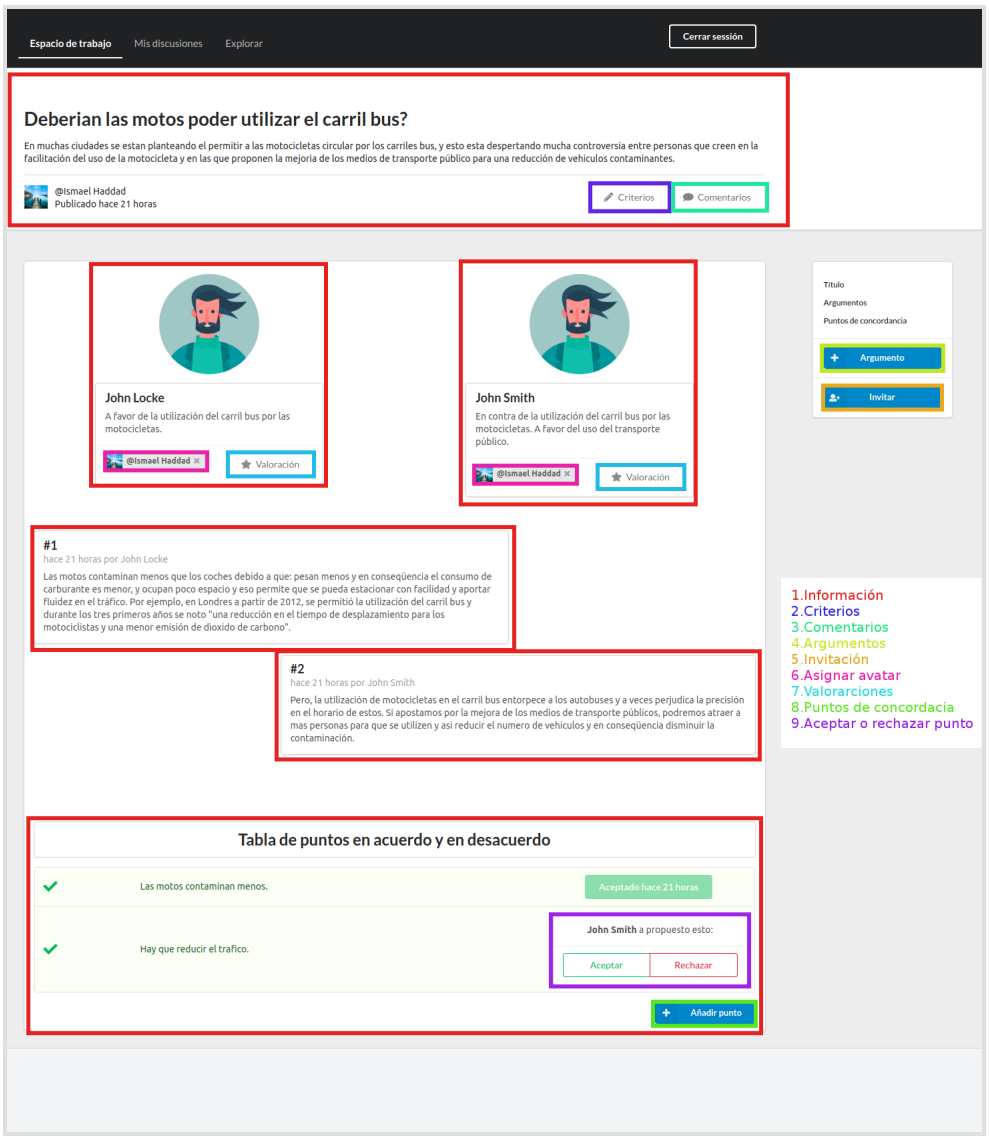


FIGURA 12.1: Pantalla de discusión

Se separa en 9 funcionalidades, los cuales se explican en los siguientes apartados.

12.1.1. Información sobre el tópico y la discusión

Los rectángulos marcado en rojo, indican las diferentes secciones donde se muestra la información de la discusión. Esta el título y una breve descripción del tópico a discutir, dos avatares con sus respectivas opiniones iniciales, los argumentos y la tabla de puntos de concordancia.

12.1.2. Criterios de la discusión

En el panel de criterios, se muestran dichos criterios en una lista. Estos se añaden a través del formulario adjuntado (imagen de la izquierda de la figura 12.2).

12.1.3. Comentarios de la discusión

Aquí se muestran los comentarios de la discusión. Estos se publican a través del formulario adjuntado (imagen de la derecha de la figura 12.2).

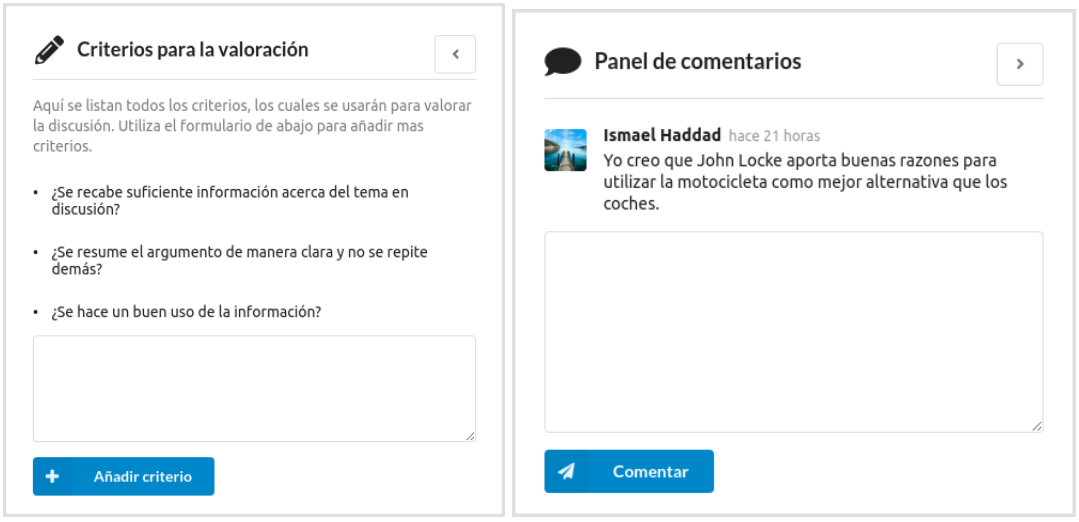


FIGURA 12.2: Panel de criterios y comentarios, respectivamente

12.1.4. Añadir un argumento

En la figura 12.3, se muestra el formulario para publicar un argumento. Hay que seleccionar el avatar quien publica el argumento (si estamos asignados a los dos) y escribir el argumento con un limite de 100 palabras.

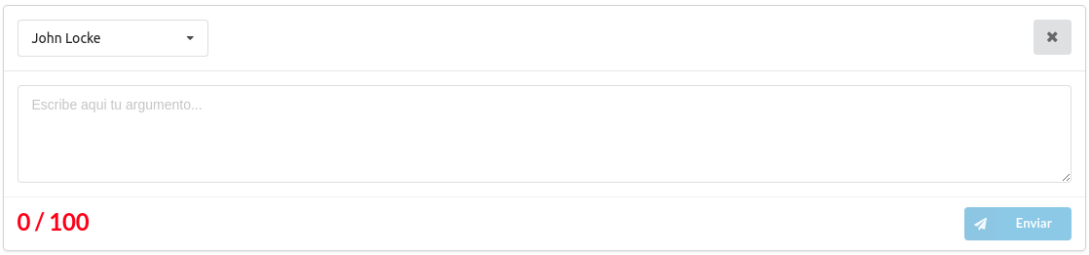


FIGURA 12.3: Formulario para publicar un argumento

12.1.5. Invitar a participar

Para invitar a un usuario a participar en la discusión, se hace mediante el formulario de *Email*. El usuario invitado recibirá un correo electrónico con un enlace hacia la discusión donde se le ha invitado (imagen de la izquierda de la figura 13.3).

12.1.6. Asignar un avatar a un participante

Para asignar un avatar a un participante, se ha de quitar la etiqueta del participante actual al avatar y seleccionar al participante correspondiente (imagen de la derecha de la figura 13.3).

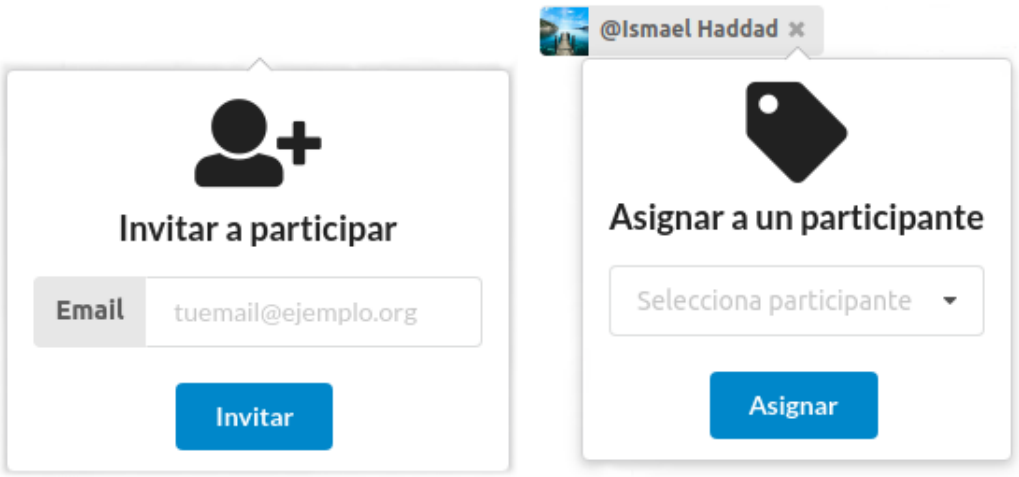


FIGURA 12.4: Diálogos para invitar y asignar a un participante, respectivamente

12.1.7. Valorar un avatar

En este panel, se listan los criterios de la discusión con las valoraciones para cada uno. Solo el propietario de la discusión puede asignar dichas valoraciones.

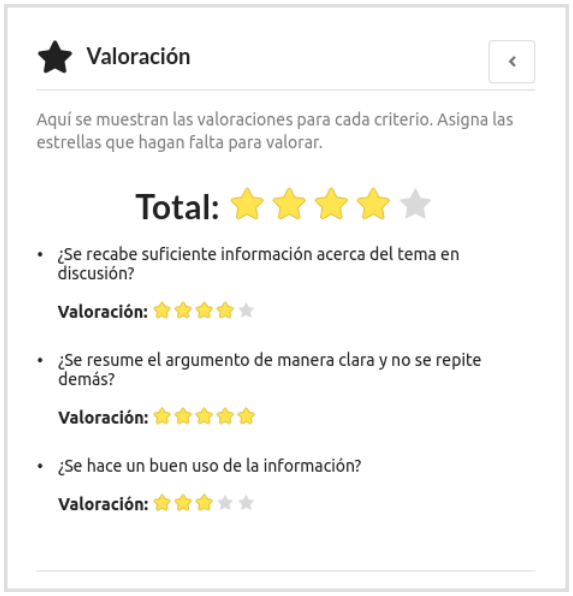


FIGURA 12.5: Panel de valoraciones

12.1.8. Proponer un punto de concordancia

Para añadir un punto de concordancia, se utiliza el siguiente formulario de la figura 12.6. Se selecciona si es un punto de acuerdo o uno de desacuerdo, el avatar que lo propone y el texto correspondiente.

FIGURA 12.6: Formulario para proponer un punto de concordancia

12.1.9. Aceptar o rechazar un punto de concordancia

Cuando un avatar propone un punto de concordancia, el otro avatar le aparecerá el siguiente cuadro de diálogo (figura 12.7) donde decidirá si quiere aceptar dicha propuesta o rechazarla.

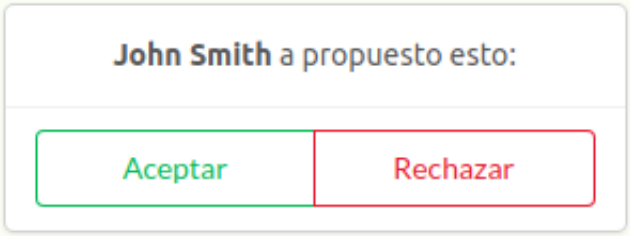


FIGURA 12.7: Diálogo para aceptar o rechazar el punto de concordancia propuesto

Capítulo 13

Caso ejemplo: Utilización como herramienta educativa

En este capítulo se describirá un escenario donde se muestra la utilización de la aplicación.

13.1. Escenario

Este caso toma como lugar un aula de un centro educativo, donde se proponen uno o varios temas para llevar una discusión entre los alumnos. El profesor describirá los criterios para la evaluación de estas discusiones y asignará una discusión a cada pareja de alumnos. Estos alumnos desarrollarán la discusión proponiendo sus argumentos durante la realización de la actividad. Finalmente, el profesor evaluará las discusiones.

13.2. Utilización de la aplicación

A continuación, se expondrá paso a paso el flujo de trabajo que se seguirá para llevar a cabo el escenario descrito anteriormente.

13.2.1. Crear y preparar una discusión

Primero, el profesor creará una discusión desde la pantalla de lista de discusiones, donde rellenará la información sobre el tópico y las opiniones que se tendrán que defender (Fig. 13.1).

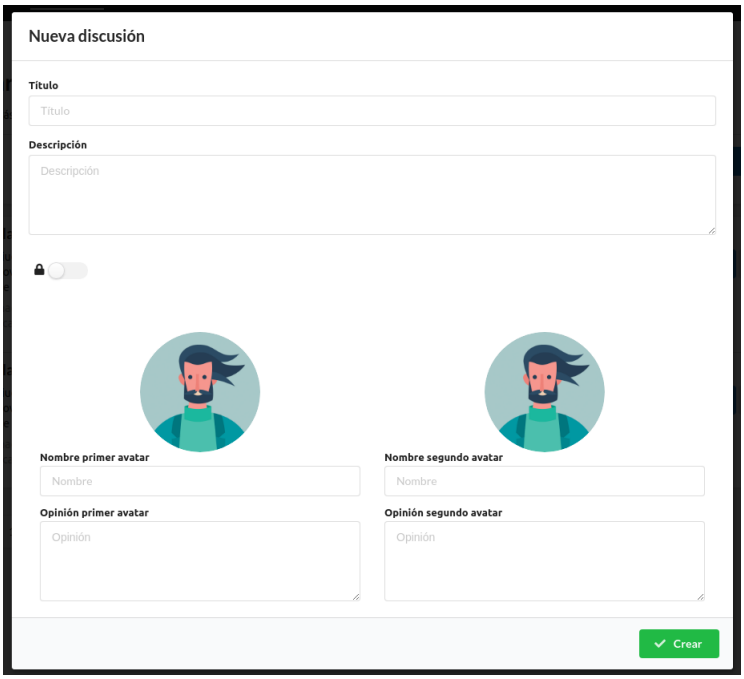


FIGURA 13.1: Diálogos para crear una nueva discusión

Seguidamente, el profesor listará los criterios de los cuales se evaluará la discusión en el panel de la izquierda (Fig. 13.2).

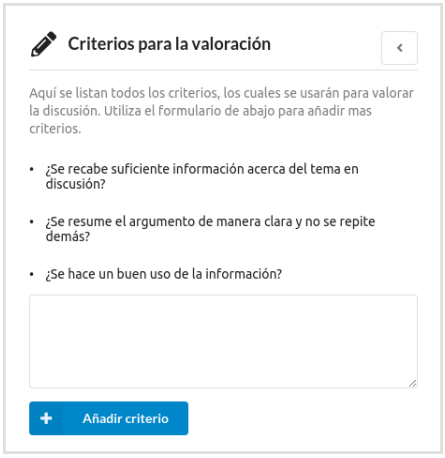


FIGURA 13.2: Panel de listado de criterios para la discusión

Finalmente, invitará a los alumnos a la discusión mediante email y asignará un avatar (o opinión) a cada uno (Fig. 13.3).

FIGURA 13.3: Diálogos para invitar y asignar a un participante, respectivamente

13.2.2. Desarrollar la discusión

Los dos alumnos, tendrán asignados uno de los dos avatares de la discusión. Cada uno de estos avatares sostienen una opinión inicial (Fig. 13.4).

FIGURA 13.4: Avatares con sus opiniones

Los alumnos tendrán que escribir bloques de argumentos con un limite de 100 palabras por bloque (Fig. 13.5).

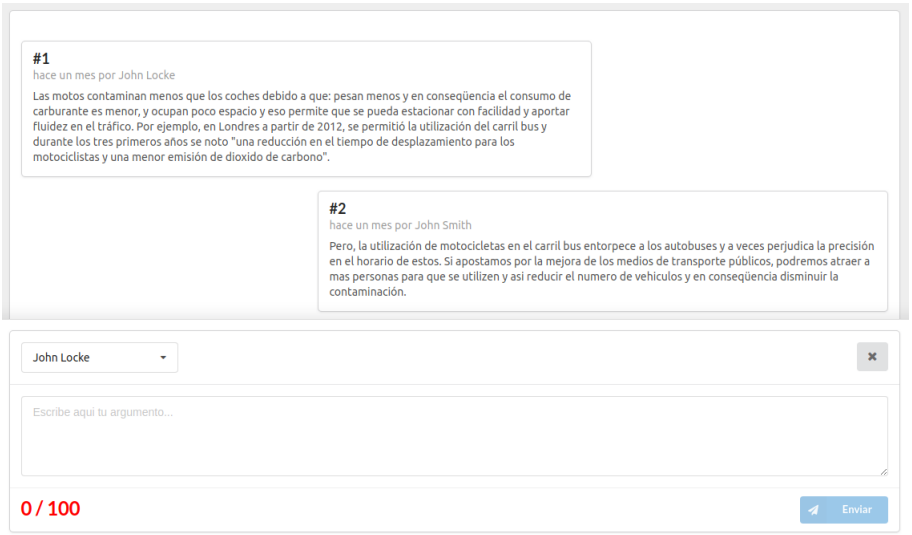


FIGURA 13.5: Argumentos de la discusión

También propondrán puntos de concordancia cuando se llegue a un acuerdo en algún punto de la discusión. Y estos, se tendrán que aceptar por el otro alumno(Fig. 13.6).

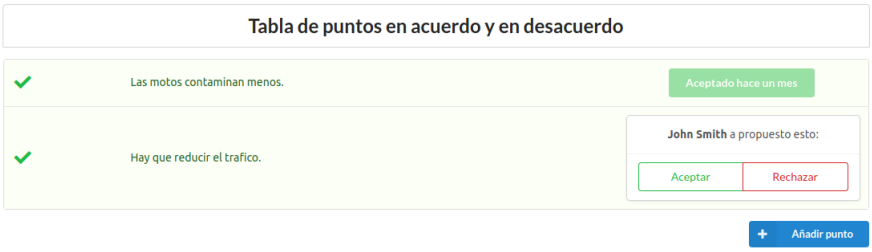


FIGURA 13.6: Tabla de puntos de concordancia

13.2.3. Evaluación de la discusión

Finalmente, el profesor valorará a cada alumno según los criterios listados al principio. También tendrá a su disposición una sección de comentarios para poder detallar y comentar lo necesario (Fig. 13.7).

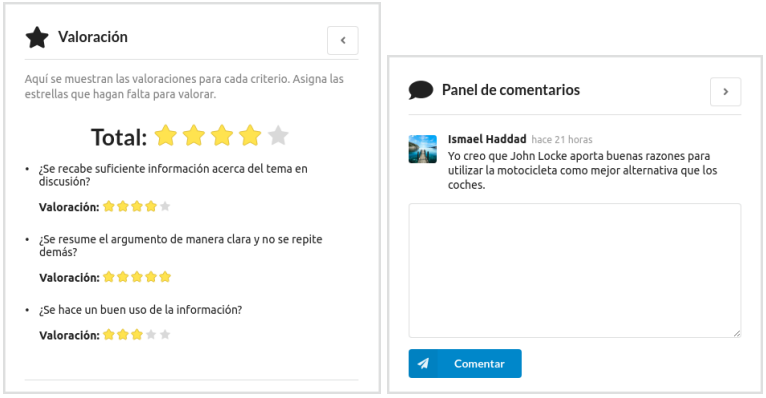


FIGURA 13.7: Valoraciones y comentarios

Capítulo 14

Conclusiones

Para concluir, procederé a exponer un resumen general de los objetivos, los conocimientos adquiridos y las competencias técnicas asimiladas.

14.1. Resumen

Al principio, se tenía como objetivo el construir una plataforma donde cualquier escritor pudiera participar en una discusión con otros usuarios y ofrecer un espació con información clara para la búsqueda sobre cualquier tópico o tema, como los que ofrecen los alojadores de código como Github.

Más adelante, han ido saliendo otros casos y escenarios en el que se podrían utilizar esta aplicación, como el ejemplo que se describe en el Capítulo 13.

Finalmente, se ha logrado satisfacer los objetivos mas importantes y prioritarios, como poder llevar a cabo una discusión o explorar varios tópicos de discusiones y se ha prescindido de pequeñas funcionalidades que complementaban las principales.

A lo largo del proyecto, se ha seguido una metodología para el desarrollo del software, en especifico la *Agile Scrum*, el cual ha facilitado realizar la construcción de la aplicación de una manera planificada y organizada. También, se ha conseguido aprender a manejar las tecnologías necesarias para cumplir con los objetivos.

14.2. Aprendizaje

Lo primero que quiero mencionar sobre lo aprendido durante el proyecto son las capacidades de las tecnologías web a día de hoy. Estas han permitido desarrollar una aplicación con la misma capacidad que una aplicación de escritorio, ofreciendo la misma interactividad y dinamismo. Además, los *frameworks web* brindan herramientas para la construcción de estos sistemas que facilitan muchísimo dicha tarea. También se ha observado que utilizar tecnologías como los *WebSockets* cambian la experiencia que se tiene con las páginas web, al introducir la capacidad de interacción en tiempo real.

Al mismo tiempo, en relación con las metodologías, se ha fijado en la importancia de las pruebas de calidad o *Tests* durante el desarrollo de la aplicación. Muchas veces, cuando se ha tenido que añadir una nueva funcionalidad, los *Tests* siempre han sido de ayuda para asegurarse de no quebrar el sistema. También han sido de ayuda durante las refactorizaciones del código. Hacer todo esto, al mismo tiempo de

mantener el buen funcionamiento del sistema, hubiera sido muy difícil de no haber sido por los *Tests*.

Por otro lado, se ha aprendido diferentes aspectos importantes sobre la gestión de proyectos, del las cuales quiero destacar la priorización de las diferentes características que debe tener el producto y calcular el valor de cada una. Ser consciente de que unas pocas de las funcionalidades que se ofrecerán aportarán el mayor valor y el resto añadirán poco.

Finalmente, quiero señalar que hay una extraña obsesión en aprender nuevas tecnologías por el simple hecho de aprenderlas. Y tendemos a enfocar dicho aprendizaje como el fin y no como el medio para un fin. Con este proyecto se ha notado que, cuando hay un objetivo bien marcado de por medio, la velocidad en la que se aprende cualquier tecnología es rápida. Se ha ido cogiendo las tecnologías sobre la marcha, indagando a través de documentaciones y buscando información para casos específicos, y se ha ido adquiriendo la gran vista generalizada de las cosas a cada paso. Todo esto hace que llegue a la conclusión de que el objetivo de un ingeniero de software debe ser conseguir la intuición de coger cualquier herramienta para poder solucionar cualquier problema que se plantee. Así, uno no se preocupa de quedarse obsoleto por la aparición de nuevas tecnologías.

14.3. Integración de conocimientos

Para la planificación y la metodología se aplican los conocimientos adquiridos en GPS (Gestión de Proyectos de Software) y PES (Projecte d'Enginyeria del Software). Se aplican métodos de trabajo Agile, Scrum en específico, que consiste en construir un Backlog y separar las diferentes fases de desarrollo de cada funcionalidad en definidos, por hacer, hechos y aceptados.

Para el análisis de requisitos, se aplican conocimientos de ER (Enginyeria de Requisits). Se toman en cuenta los stakeholder para identificar roles y necesidades. Se definen los requisitos del sistema y se especifican los datos y el comportamiento del mismo.

Para el diseño del software, se aplican conocimientos de AS (Arquitectura de Software), ASW (Aplicaciones y Servicios Web) y DBD (Diseño de Bases de Datos). Se usan los patrones estudiados para construir la arquitectura del sistema, como la división de responsabilidades en capas, estándares para la comunicación entre servicios y las buenas prácticas para construir la base de datos.

14.4. Competencias técnicas

CES1.2 Donar solució a problemes d'integració en funció de les estratègies, dels estàndards i de les tecnologies disponibles. [Bastante]

Para construir el sistema se ha utilizado tecnologías y servicios existentes, como los *hosts PaaS* [1], *Heroku* [14] y *Netlify* [15], para alojar la aplicación, utilizar *Redis* para el

[1] Platform as a Service

funcionamiento de los *WebSockets* [6] y integraciones de servicios como la autenticación mediante *Google* [36] y el servicio de envio de emails *SendGrid* [37]. También hay que señalar la integración entre *Frontend* y *Backend*, aplicando estándares de comunicación como la *RESTful API*.

CES1.5: Especificar, dissenyar, implementar i avaluar bases de dades. [Un poco]

A la hora de diseñar la base de datos de la aplicación, se han tomado en cuenta varias nociones básicas, sobre todo la de especificación y diseño, como construir las tablas de cierta manera para optimizar el uso que se les vaya a dar. Y para la implementación se ha utilizado un ORM [2] que ofrece el *framework Ruby on Rails* [33], que nos facilita la configuración de la base de datos y brinda una capa de abstracción para poder cambiar entre diferentes bases de datos fácilmente.

CES1.7: Controlar la qualitat i dissenyar proves en la producció de software. [Bastante]

En este aspecto, se ha explorado varios tipos de *Tests* automáticos y se ha profundizado en las mas relevantes y acordes al proyecto. Estas pruebas de calidad automáticas forman parte de la metodología utilizada durante el desarrollo. La metodología consistía en definir las pruebas extraídas de los requisitos para el sistema antes de implementar la funcionalidad y luego añadir la pruebas necesarias según las peculiaridades de cada una.

CES1.9: Demostrar comprensió en la gestió i govern dels sistemes software. [Bastante]

A parte de diseñar el sistema, se ha tenido que coordinar este entre sus dos partes, *Frontend* y *Backend*, estableciendo un patrón para comunicar dos tecnologías diferentes. También se ha tenido que integrar distintos sistemas como la autenticación mediante *Google* [36], el servicio de envio de emails *SendGrid* [37] y otros más. En cuanto a los fallos del sistema, la gran mayoría de veces, se han podido detectar y corregir errores que iban surgiendo fácilmente gracias al uso de *debuggers*.

CES2.1: Definir i gestionar els requisits d'un sistema software. [Bastante]

Al principio se ha ido recopilando los diversos requisitos en base a los objetivos que teníamos para el proyecto. Durante el proyecto, mediante *Pivotal Tracker* [10], se han ido priorizando y discutiendo las funcionalidades con el director de la empresa en las reuniones de retrospectiva para cada iteración.

14.5. Trabajo futuro

Como se ha visto en el capítulo 5, en el apartado de cambios realizados a la planificación, quedan algunas funcionalidades, no esenciales pero si buenos de tener, que

[2] Object-Relational Mapping

no se han implementado. De estas, me gustaría priorizar las herramientas de apoyo a la hora de argumentar. Hasta ahora, se ha implementado el panel de criterios, con la idea de ayudar a mejorar la dirección de la discusión. Como trabajo futuro, es deseable realizar un estudio de como se podría facilitar el procesos de escribir un argumento y implementar una solución efectiva.

Otras funcionalidades extras que serían buenos de tener son: el sistema de citas y relacionar bloques de argumentos como respuestas y contra-argumentos. Todo esto complementaria el sistema ya construido para ofrecer mas libertad a la hora de utilizar la aplicación.

14.6. Otros proyectos trabajados durante las prácticas externas

Las 145 horas restantes de las 735 de las prácticas externas se ha dedicado en la construcción de una nueva versión de la página web de la empresa utilizando tecnologías como *Vue* y el *framework Nuxt js*, que nos permite crear páginas compatibles con el SEO[3] y usar componentes como los vistos en *React js* de manera muy sencilla. Todo esto, integrando un *headless CMS*, como *Contentful*, para gestionar el contenido de la página web. Estos *headless CMS* son una nueva forma de gestionar contenido de las páginas web a través de interfaces REST API.

[3]Search Engine Optimization (Optimización de motor de busqueda)

Capítulo 15

Leyes y regulaciones

A continuación se exponen las regulaciones y leyes que afectan al proyecto.

15.1. Licencias

En la siguiente tabla se listan las diferentes librerías y software utilizado con el tipo de licencias que utilizan.

MIT License	
React js	Copyright (c) Facebook, Inc. and its affiliates.
Semantic UI	
axios	Copyright (c) 2014-present Matt Zabriskie
moment js	Copyright (c) JS Foundation and other
actioncable	Copyright (c) 2015-2019 Basecamp, LLC
RubyonRails	Copyright (c) 2004-2019 David Heinemeier Hansson
jbuilder	Copyright (c) 2011-2018 David Heinemeier Hansson, 37signals
Kaminarion	Copyright (c) 2011 Akira Matsuda
Minitest	Copyright (c) 2019 Mike Moore
PostgreSQL License	
PostgreSQL	Portions Copyright © 1996-2019, The PostgreSQL Global Development Group. Portions Copyright © 1994, The Regents of the University of California.
BSD License	
Redis	
ISC License	
react-grid-system	Copyright (c) 2016, JSxMachina
Apache License 2.0	
google-id-token	Copyright Google
Google Terms[1]	
Google SignIn	Copyright Google

CUADRO 15.1: Tabla de licencias de uso de software

15.2. Leyes

A lo referente a leyes, hay que mencionar la *Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales*, ya que los datos utilizados para identificar al usuario están regulados por dicha ley.

Índice de figuras

Índice de cuadros

Bibliografía

[1] *Prototyp SE*. URL: http://www.prototyp.se/en (visitado 20-03-2019).

[2] Jennifer A. Moon. *Critical thinking: an exploration of theory and practice*. Routledge, 2008.

[3] Brooke Noel Moore y Richard Parker. *Critical thinking*. McGraw-Hill Education, 2017.

[4] Stephen Toulmin, Richard Rieke y Allan Janik. *An introduction to reasoning*. Macmillan, 2002.

[5] Benilde García Cabrero y Vania Jocelyn Pineda Ortega. «La construcción de conocimiento en foros virtuales de discusión entre pares». En: *Revista mexicana de investigación educativa* (mar. de 2010), págs. 85 -111. ISSN: 1405-6666. URL: http://www.scielo.org.mx/scielo.php?script=sci_arttext&pid=S1405-66662010000100006&nrm=iso.

[6] I. Fette. *The WebSocket Protocol*. 2011. URL: https://tools.ietf.org/html/rfc6455 (visitado 20-03-2019).

[7] Gary Pike y Kathryne Drezek McConnell. «The Dependability of VALUE Scores: Lessons Learned and Future Directions». En: *Association of American Colleges Universities* (feb. de 2019). URL: https://www.aacu.org/peerreview/2018/Fall/Research.

[8] Github.com. *Build Software Better, Together*. URL: http://www.github.com (visitado 20-03-2019).

[9] Bitbucket.com. *The Git Solution for Professional Teams*. URL: http://www.bitbucket.com (visitado 20-03-2019).

[10] Pivotal Tracker. *Agile Project Management*. URL: https://pivotaltracker.com (visitado 20-03-2019).

[11] Plan IT Poker. *Online Scrum planning poker for Agile project teams*. URL: http://www.planitpoker.com (visitado 20-03-2019).

[12] Google. *Google Drive*. URL: https://www.google.es/drive/apps.html (visitado 20-03-2019).

[13] Travis CI. *Test and Deploy Your Code with Confidence*. URL: https://travis-ci.com (visitado 20-03-2019).

[14] Heroku. *Cloud Application Platform*. URL: https://www.heroku.com/home (visitado 20-03-2019).

[15] Netlify. *All-in-one platform for automating modern web projects*. URL: https://www.netlify.com (visitado 20-03-2019).

[16] Git. *–everything-is-local*. URL: https://git-scm.com (visitado 20-03-2019).

[17] MiniTest-Rails. *minitest-rails*. URL: http://blowmage.com/minitest-rails (visitado 20-03-2019).

[18] TeamGantt. *Intuitive and Beautiful Project Planning*. URL: https://www.teamgantt.com (visitado 20-03-2019).

[19] Toshiba. *Satellite P50-B-10V*. URL: https://www.toshiba.es/discontinued-products/satellite-p50-b-10v/?categoryNameProductPage=laptops (visitado 20-03-2019).

[20] Ubuntu. *Ubuntu Desktop for Developers*. URL: https://www.ubuntu.com/desktop/developers (visitado 20-03-2019).

[21] Linux. *Linux.org*. URL: https://www.linux.org (visitado 20-03-2019).

[22] Google Search. *Navegador Web Google Chrome*. URL: https://www.google.com/intl/es_ALL/chrome (visitado 20-03-2019).

[23] Atom. *A Hackable Text Editor for the 21st Century*. URL: https://www.atom.io (visitado 20-03-2019).

[24] Postman. *Postman Simplifies API Development*. URL: https://www.getpostman.com (visitado 20-03-2019).

[25] Michael Page. *Estudios De Remuneración*. URL: https://www.michaelpage.es/prensa-estudios/estudios/estudios-de-remuneracion (visitado 20-03-2019).

[26] Selectra Espa. *Precio KWh 2019: ¿Qué Compañía Ofrece La Luz Más Barata?* Ene. de 2019. URL: https://www.michaelpage.es/prensa-estudios/estudios/estudios-de-remuneracion (visitado 20-03-2019).

[27] James Suzanne Robertson. *Volere Requirements Specification Template*. 2017. URL: https://www.volere.org/ (visitado 20-03-2019).

[28] Auth0. *Single Page Login Flow*. URL: https://auth0.com/docs/flows/concepts/single-page-login-flow (visitado 20-03-2019).

[29] Software Engineering Stack Exchange. *How to architecture a realtime-heavy websockets-based web application?* URL: https://softwareengineering.stackexchange.com/questions/327644/how-to-architecture-a-realtime-heavy-websockets-based-web-application (visitado 20-03-2019).

[30] React JS. *A JavaScript library for building user interfaces*. URL: https://reactjs.org (visitado 20-03-2019).

[31] Mozilla. *¿Qué es JavaScript?* URL: https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/QuÃl'_es_JavaScript (visitado 20-03-2019).

[32] Semantic UI. *User Interface is the language of the web*. URL: https://semantic-ui.com (visitado 20-03-2019).

[33] Ruby on Rails. *Imagine what you could build if you learned Ruby on Rails...* URL: https://rubyonrails.org (visitado 20-03-2019).

[34] Ruby Language. *Un lenguaje de programación dinámico y de código abierto enfocado en la simplicidad y productividad*. URL: https://www.ruby-lang.org/es/ (visitado 20-03-2019).

[35] Rails Guides. *Action Cable Overview*. URL: https://guides.rubyonrails.org/action_cable_overview.html (visitado 20-03-2019).

[36] Google. *Google Sign-In for Websites*. URL: https://developers.google.com/identity/sign-in/web (visitado 20-03-2019).

[37] SendGrid. *Send Email With Confidence*. URL: https://sendgrid.com (visitado 20-03-2019).

[38] Vincent Driessen. *A successful Git branching model*. URL: https://nvie.com/posts/a-successful-git-branching-model/ (visitado 20-03-2019).

[39] Atlassian. *Gitflow Workflow*. URL: https://es.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow (visitado 20-03-2019).

Apéndice A

Documentación de peticiones REST API

A.1. Login

Identificar al usuario en el sistema mediante el `id_token` de Google Sign In y obtener un `session_token` para poder hacer peticiones que requieran de autorización.

Petición

Método	URL
POST	/tokensignin

Tipo	Parámetro	Valor
HEAD	Authorization	string

- `Authorization`. Debe ser enviada en todas las peticiones que necesiten de autorización.

Respuesta

200

```
{
    "userId": 1,
    "name": "Ismael Haddad",
    "sessionToken": <auth_key>,
    "sessionTokenExpiresAt": "2019-04-09 12:10:23"
}
```

`auth_key (string)` - todas las llamadas que necesiten de autorización tendrán que llevar la key en el header

401

```
{"error":"Validation failed."}
```

500

```
{"error":"Internal server error."}
```

A.2. Obtener una discusión

Obtener una discusión identificada por `id`.

Petición

Método	URL
GET	/discussions/<discussion_id>

Tipo	Parámetro	Valor
URL_PARAM	<discussion_id>	number

- `discussion_id`. Identificador de la discusión que se quiere obtener.

Respuesta

200

```
{
    "id": 1,
    "topicTitle": "El titulo del topico a discutir",
    "topicDescription": "Descripcion del topico a discutir",
    "publishTime": "2019-04-09 12:10:23",
    "forkedFrom": null,
    "private": false,
    "owner": {
        "id": 1,
        "name": "Ismael Haddad",
        "imageUrl": "http:/example.com/image.png",
    },
    "avatarOne": {
        "id": 1,
        "name": "John",
        "opinion": "Opinion del primer avatar",
        "assignedToUserId": 1,
    },
    "avatarTwo": {
        "id": 1,
        "name": "Jack",
        "opinion": "Opinion del segundo avatar",
        "assignedToUserId": 1,
    },
    "participantsId": [1,2]
}
```

404

```
{"error":"Couldn't find Discussion with id= <discussion_id
   >."}
```

500

```
{"error":"Internal server error."}
```

A.3. Obtener los argumentos de una discusión

Obtener los argumentos de una discusión identificada por `id`.

Petición

Método	URL
GET	/discussions/<discussion_id>/arguments

Tipo	Parámetro	Valor
URL_PARAM	<discussion_id>	number

- `discussion_id`. Identificador de la discusión que se quiere obtener.

Respuesta

200

Un *array* de todos los argumentos propuestos en la discusión.

```
[
    {
        "id": 1,
        "num": 1,
        "content": "Contenido del argumento",
        "publishTime": "2019-04-09 12:10:23",
        "fromAvatar": {
            "id": 1,
            "name": "John Locke",
        },
    },
]
```

404

```
{"error":"Couldn't find Discussion with id= <discussion_id
   >."}
```

500

```
{"error":"Internal server error."}
```

A.4. Obtener los puntos de concordancia de una discusión

Obtener los puntos de concordancia de una discusión identificada por `id`.

Petición

Método	URL
GET	/discussions/<discussion_id>/agreements

Tipo	Parámetro	Valor
URL_PARAM	<discussion_id>	number

- `discussion_id`. Identificador de la discusión que se quiere obtener.

Respuesta

200

Un *array* de todos los puntos propuestos en la discusión.

```
[
    {
        "id": 1,
        "content": "Contenido del punto de concordancia",
        "isAgree": true,
        "isAccepted": true,
        "acceptedAt": "2019-04-09 12:10:23",
        "proposedByAvatar": {
            "id": 1,
            "name": "John Locke",
        },
    },
]
```

404

```
{"error":"Couldn't find Discussion with id= <discussion_id
   >."}
```

500

```
{"error":"Internal server error."}
```

A.5. Proponer un argumento en una discusión

Proponer un argumento en una discusión identificada por `id`.

Petición

Método	URL
POST	/discussions/<discussion_id>/arguments

Tipo	Parámetro	Valor
HEAD	Authorization	string
URL_PARAM	<discussion_id>	number
POST_BODY	<avatar_id>	number
POST_BODY	<content>	string

- `Authorization`. Debe ser enviada en el header de la petición para identificar al usuario.
- `discussion_id`. Identificador de la discusión donde se propone el argumento.
- `avatar_id`. Identificador del avatar que propone el argumento.
- `content`. Contenido del argumento.

Respuesta

200

```
{
    "id": 1,
    "num": 1,
    "content": "Contenido del argumento",
    "publishTime": "2019-04-09 12:10:23",
    "fromAvatar": {
        "id": 1,
        "name": "John Locke",
    },
}
```

401

```
{"error":"session_token expired or doesn't exists."}
```

422

```
{"error":"Invalid data."}
```

500

```
{"error":"Internal server error."}
```

A.6. Proponer un punto de concordancia en una discusión

Proponer un punto de concordancia en una discusión identificada por `id`.

Petición

Método	URL
POST	/discussions/<discussion_id>/agreements

Tipo	Parámetro	Valor
HEAD	Authorization	string
URL_PARAM	<discussion_id>	number
POST_BODY	<avatar_id>	number
POST_BODY	<content>	string
POST_BODY	<is_agree>	boolean

- `Authorization`. Debe ser enviada en el header de la petición para identificar al usuario.
- `discussion_id`. Identificador de la discusión donde se propone el punto.
- `avatar_id`. Identificador del avatar que propone el punto.
- `content`. Contenido del punto de concordancia o discrepancia.
- `is_agree`. `[true]` si se trata de un punto en acuerdo. `[false]` si se trata de un punto en desacuerdo.

Respuesta

200

```
{
    "id": 1,
    "content": "Contenido del punto de concordancia",
    "isAgree": true,
    "isAccepted": false,
    "acceptedAt": null,
    "proposedByAvatar": {
        "id": 1,
        "name": "John Locke",
    },
}
```

401

```
{"error":"session_token expired or doesn't exists."}
```

422

```
{"error":"Invalid data."}
```

500

```
{"error":"Internal server error."}
```

A.7. Aceptar o rechazar un punto de concordancia de una discusión

Aceptar o rechazar un punto de concordancia identificada por `agreement_id` en una discusión identificada por `discussion_id`.

Petición

Método	URL
PUT	/discussions/<discussion_id>/agreements /<agreement_id>

Tipo	Parámetro	Valor
HEAD	Authorization	string
URL_PARAM	<discussion_id>	number
URL_PARAM	<agreement_id>	number
PUT_BODY	<avatar_id>	number
PUT_BODY	<is_accepted>	boolean

- `Authorization`. Debe ser enviada en el header de la petición para identificar al usuario.
- `discussion_id`. Identificador de la discusión donde se propone el punto.
- `agreement_id`. Identificador del punto de concordancia.

- `avatar_id`. Identificador del avatar que acepta o rechaza el punto.
- `is_agree`. `[true]` si se acepta el punto. `[false]` si se rechaza el punto.

Respuesta

200
Si se acepta el punto:

```
{
    "id": 1,
    "content": "Contenido del punto de concordancia",
    "isAgree": true,
    "isAccepted": true,
    "acceptedAt": "2019-04-09 12:10:23",
    "proposedByAvatar": {
        "id": 1,
        "name": "John Locke",
    },
}
```

Si se rechaza el punto:

```
{"message":"Agreement rejected and deleted."}
```

401

```
{"error":"session_token expired or doesn't exists."}
```

422

```
{"error":"Invalid data."}
```

500

```
{"error":"Internal server error."}
```

A.8. Invitar a una persona para participar en una discusión

Invitar, mediante email, a una persona para participar en una discusión identificada por `discussion_id`.

Petición

Método	URL
POST	/discussions/<discussion_id>/invite

Tipo	Parámetro	Valor
HEAD	Authorization	string
URL_PARAM	<discussion_id>	number
POST_BODY	<email>	string

- `Authorization`. Debe ser enviada en el header de la petición para identificar al usuario.

- `discussion_id`. Identificador de la discusión donde se invita a participar.
- `email`. Correo electrónico de la persona invitada.

Respuesta

200

```
1 {"message": "Invitation send."}
```

401

```
1 {"error":"Only owner of the discussion can make invitation."}
```

422

```
1 {"error":"Couldn't generate an invitation."}
```

500

```
1 {"error":"Internal server error."}
```

A.9. Verificar una invitación para participar en una discusión

Verificar una invitación identificado por token para participar en una discusión identificada por `discussion_id`.

Petición

Método	URL
PUT	/discussions/<discussion_id>/verify_invitation

Tipo	Parámetro	Valor
HEAD	Authorization	string
URL_PARAM	<discussion_id>	number
PUT_BODY	<token>	string

- `Authorization`. Debe ser enviada en el header de la petición para identificar al usuario.
- `discussion_id`. Identificador de la discusión donde se invita a participar.
- `token`. Identificador para verificar la invitación.

Respuesta

200

```
1 {"message": "Invitation verified."}
```

401

```
1 {"error":"session_token expired or doesn't exists."}
```

403

```
1 {"error":"Token already verified."}
```

404

```
1 {"error":"Couldn't find token invitation."}
```

422

```
1 {"error":"Couldn't verify the invitation."}
```

500

```
1 {"error":"Internal server error."}
```

A.10. Asignar un avatar a un participante en una discusión

Asignar un avatar identificado por `avatar_id` a un participante identificado por `user_id` en una discusión identificada por `discussion_id`.

Petición

Método	URL
PUT	/discussions/<discussion_id>/avatars/<avatar_id>

Tipo	Parámetro	Valor
HEAD	Authorization	string
URL_PARAM	<discussion_id>	number
URL_PARAM	<avatar_id>	number
PUT_BODY	<user_id>	number

- `Authorization`. Debe ser enviada en el header de la petición para identificar al usuario.
- `discussion_id`. Identificador de la discusión.
- `avatar_id`. Identificador del avatar de la discusión.
- `user_id`. Identificador del participante que se asigna al avatar de la discusión.

Respuesta

200

```
1 {"message": "Avatar with id=<avatar_id> is assigned to user
    with id=<user_id>."}
```

401

```
1 {"error":"session_token expired or doesn't exists."}
```

403

```
1 {"error":"User isn't participating in the discussion."}
```

404

```
1 {"error":"User doesn't exists."}
```

500

```
1 {"error":"Internal server error."}
```

A.11. Obtener un usuario

Obtener un usuario identificado por `user_id`.

Petición

Método	URL
GET	/users/<user_id>

Tipo	Parámetro	Valor
URL_PARAM	<user_id>	number

- `user_id`. Identificador del usuario que se quiere obtener.

Respuesta

200

```
{
    "id": 1,
    "name": "Ismael Haddad",
    "imageUrl": "http:/example.com/profileImage.png"
}
```

404

```
{"error":"Couldn't find User with id=<user_id>."}
```

500

```
{"error":"Internal server error."}
```

A.12. Obtener lista de discusiones

Obtener una lista de discusiones paginada.

Petición

Método	URL
GET	/discussions?page=<page>&user_id=<user_id>

Tipo	Parámetro	Valor
HEAD	Authorization [optional]	string
URL_PARAM	<page>	number
URL_PARAM	<user_id> [optional]	number

- `Authorization`. Debe ser enviada en el header de la petición para identificar al usuario.
- `page`. Página que se quiere obtener.
- `user_id`. Identificador del usuario del cual queremos coger las discusiones.

Respuesta

200

Un *array* de todos los argumentos propuestos en la discusión.

```
{
    discussions: [
        {
            "id": 1,
            "topicTitle": "El titulo del topico a discutir",
            "topicDescription": "Descripcion del topico",
            "publishTime": "2019-04-09 12:10:23",
            "forkedFrom": null,
            "private": false,
            "owner": {
                "id": 1,
                "name": "Ismael Haddad",
                "imageUrl": "http:/example.com/image.png",
            }
        },
    ],
    "pages": {
        "current": 1,
        "total": 50,
    }
}
```

401

```
{"error":"session_token expired or doesn't exists."}
```

404

```
{"error":"Couldn't find User with id=<user_id>."}
```

500

```
{"error":"Internal server error."}
```

A.13. Crear una nueva discusión

Crear una nueva discusión.

Petición

Método	URL
POST	/discussions

Tipo	Parámetro	Valor
HEAD	Authorization [optional]	string
POST_BODY	<topic_title>	string
POST_BODY	<topic_description>	string
POST_BODY	<name_avatar_one>	string
POST_BODY	<opinion_avatar_one>	string
POST_BODY	<name_avatar_two>	string
POST_BODY	<opinion_avatar_two>	string
POST_BODY	<private>	boolean

- `Authorization`. Debe ser enviada en el header de la petición para identificar al usuario.
- `topic_title`. El título del tópico de la discusión.
- `topic_description`. La descripción del tópico de la discusión.
- `name_avatar_one` y `name_avatar_two`. Los nombres de los dos avatares.
- `opinion_avatar_one` y `opinion_avatar_two`. Las dos posiciones iniciales de cada avatar.
- `page`. `[true]` si es una discusión privada. `[false]` si es una discusión pública.

Respuesta

200

```
{
    "id": 1,
    "topicTitle": "El titulo del topico a discutir",
    "topicDescription": "Descripcion del topico a discutir",
    "publishTime": "2019-04-09 12:10:23",
    "forkedFrom": null,
    "private": false,
    "owner": {
        "id": 1,
        "name": "Ismael Haddad",
        "imageUrl": "http:/example.com/image.png",
    },
    "avatarOne": {
        "id": 1,
        "name": "John",
        "opinion": "Opinion del primer avatar",
        "assignedToUserId": 1,
    },
    "avatarTwo": {
        "id": 1,
        "name": "Jack",
        "opinion": "Opinion del segundo avatar",
        "assignedToUserId": 1,
    },
    "participantsId": [1]
```

```
26 }
```

401

```
1 {"error":"session_token expired or doesn't exists."}
```

422

```
1 {"error":"Invalid data."}
```

500

```
1 {"error":"Internal server error."}
```

A.14. Borrar una discusión propia

Borrar una discusión propia identificada por `id`.

Petición

Método	URL
DELETE	/discussions/<discussion_id>

Tipo	Parámetro	Valor
HEAD	Authorization [optional]	string
URL_PARAM	<discussion_id>	number

- `Authorization`. Debe ser enviada en el header de la petición para identificar al usuario.
- `discussion_id`. Identificador de la discusión que se quiere borrar.

Respuesta

200

```
1 {"message": "Discussion deleted"}
```

401

```
1 {"error":"session_token expired or doesn't exists."}
```

403

```
1 {"error":"Only owner of the discussion can delete it."}
```

404

```
1 {"error":"Couldn't find Discussion with id= <discussion_id
    >."}
```

500

```
1 {"error":"Internal server error."}
```

A.15. Hacer *fork* de una discusión

Hacer *fork* de una discusión identificada por `id`.

Petición

Método	URL
PUT	/discussions/<discussion_id>/fork

Tipo	Parámetro	Valor
HEAD	Authorization [optional]	string
URL_PARAM	<discussion_id>	number

- `Authorization`. Debe ser enviada en el header de la petición para identificar al usuario.
- `discussion_id`. Identificador de la discusión al cual se va a hacer *fork*.

Respuesta

200

```
{"message": "Forked with success.","discussionId": 2}
```

401

```
{"error":"session_token expired or doesn't exists."}
```

404

```
{"error":"Couldn't find Discussion with id= <discussion_id
   >."}
```

422

```
{"error":"Invalid fork."}
```

500

```
{"error":"Internal server error."}
```

A.16. Obtener los comentarios de una discusión

Obtener los comentarios de una discusión.

Petición

Método	URL
GET	/discussions/<discussion_id>/general_comments

Tipo	Parámetro	Valor
URL_PARAM	<discussion_id>	number

- `discussion_id`. Identificador de la discusión del cual se quiere obtener los comentarios.

Respuesta

200

Un *array* de todos los comentarios de la discusión.

```
[
    {
        "id": 1,
        "text": "Esto es un comentario",
        "date": "2019-04-29T12:46:10.807Z",
        "user":{
            "id": 1,
            "name": "Ismael Haddad",
            "imageUrl": "http:/example.com/image.png"
        }
    }, ...
]
```

404

```
{"error":"Couldn't find Discussion with id=<discussion_id>."}
```

500

```
{"error":"Internal server error."}
```

A.17. Publicar un comentario en una discusión

Publicar un comentario en una discusión.

Petición

Método	URL
POST	/discussions/<discussion_id>/general_comments

Tipo	Parámetro	Valor
HEAD	Authorization	string
URL_PARAM	<discussion_id>	number
POST_BODY	<text>	string

- `Authorization`. Debe ser enviada en el header de la petición para identificar al usuario.
- `discussion_id`. Identificador de la discusión en el cual se publica el comentario.
- `text`. Contenido del comentario.

Respuesta

200

```
{
    "id": 1,
    "text": "Esto es un comentario",
    "date": "2019-04-29T12:46:10.807Z",
    "user":{
        "id": 1,
        "name": "Ismael Haddad",
        "imageUrl": "http:/example.com/image.png"
    }
}
```

401

```
{"error":"session_token expired or doesn't exists."}
```

404

```
{"error":"Couldn't find Discussion with id=<discussion_id>."}
```

500

```
{"error":"Internal server error."}
```

A.18. Obtener lista de criterios de una discusión

Obtener lista de criterios de una discusión.

Petición

Método	URL
GET	/discussions/<discussion_id>/criteria

Tipo	Parámetro	Valor
URL_PARAM	<discussion_id>	number

- discussion_id. Identificador de la discusión del cual se quiere obtener los criterios.

Respuesta

200

Un *array* de todos los criterios de la discusión.

```
[
    {
        "id": 1,
        "text": "Se recabe suficiente informacion acerca del
           tema en discusion?"
    },
    {
        "id": 2,
        "text": "Se resume el argumento de manera clara y no
           se repite demas?"
```

```
    },
    {
        "id": 3,
        "text": "Se hace un buen uso de la informacion?"
    }
]
```

404

```
{"error":"Couldn't find Discussion with id=<discussion_id>."}
```

500

```
{"error":"Internal server error."}
```

A.19. Añadir un criterio para una discusión

Añadir un criterio para una discusión identificada por `id`.

Petición

Método	URL
POST	/discussions/<discussion_id>/criteria

Tipo	Parámetro	Valor
HEAD	Authorization	string
URL_PARAM	<discussion_id>	number
POST_BODY	<text>	string

- `Authorization`. Debe ser enviada en el header de la petición para identificar al usuario.
- `discussion_id`. Identificador de la discusión en el cual se añade el criterio.
- `text`. Contenido del criterio.

Respuesta

200

```
{
    "id": 1,
    "text": "Se recabe suficiente informacion acerca del tema
        en discusion?"
}
```

401

```
{"error":"session_token expired or doesn't exists."}
```

403

```
{"error":"User with id=<user_id> is not the owner of the
   discussion."}
```

404

```
{"error":"Couldn't find Discussion with id=<discussion_id>."}
```

500

```
{"error":"Internal server error."}
```

A.20. Obtener valoraciones de un avatar

Obtener valoraciones de un avatar identificado por `id`.

Petición

Método	URL
GET	/discussions/<discussion_id>/avatar/<avatar_id> /ratings

Tipo	Parámetro	Valor
URL_PARAM	<discussion_id>	number
URL_PARAM	<avatar_id>	number

- `discussion_id`. Identificador de la discusión del cual se quiere obtener el avatar.
- `avatar_id`. Identificador del avatar del cual se quiere obtener las valoraciones.

Respuesta

200

Un *array* de todas las valoraciones con los criterios de la discusión.

```
[
    {
        "id": 1,
        "rating": 4,
        "criterium":{
            "id": 1,
            "text": "Se recabe suficiente informacion acerca
               del tema en discusion?"
        }
    }, ...
]
```

403

```
{"error":"Avatar with id=<avatar_id> does not participate in
   the discussion with id=<discussion_id>."}
```

404

```
{"error":"Couldn't find Discussion with id=<discussion_id>."}
```

500

```
{"error":"Internal server error."}
```

A.21. Valorar un avatar según criterio para una discusión

Valorar un avatar según criterio para una discusión identificada por `id`.

Petición

Método	URL
GET	/discussions/<discussion_id>/avatar/<avatar_id> /ratings

Tipo	Parámetro	Valor
HEAD	Authorization	string
URL_PARAM	<discussion_id>	number
URL_PARAM	<avatar_id>	number
POST_BODY	<rating>	number

- `Authorization`. Debe ser enviada en el header de la petición para identificar al usuario.
- `discussion_id`. Identificador de la discusión del cual se valora al avatar.
- `avatar_id`. Identificador del avatar al cual se valora.
- `rating`. Valoración del 0 al 5.

Respuesta

200

```
{
    "id": 1,
    "rating": 4,
    "criterium":{
        "id": 1,
        "text": "Se recabe suficiente informacion acerca del
           tema en discusion?"
    }
}
```

401

```
{"error":"session_token expired or doesn't exists."}
```

403

```
{"error":"Avatar with id=<avatar_id> does not participate in
   the discussion with id=<discussion_id>."}
```

403

```
{"error":"User with id=<user_id> is not the owner of the
   discussion with id=<discussion_id>."}
```

404

```
{"error":"Couldn't find Discussion with id=<discussion_id>."}
```

422

```
{"error":"Invalid data."}
```

500

```
{"error":"Internal server error."}
```